BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

USING COUNTER-EXAMPLES IN CONTROLLER SYNTHESIS FOR POMDPS

VYUŽITÍ PROTIPŘÍKLADŮ V SYNTÉZE KONTROLERŮ PRO POMDP

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. JAKUB FREJLACH**
AUTOR PRÁCE

SUPERVISOR **doc. RNDr. MILAN ČEŠKA, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2023

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Master's Thesis Assignment

147446

Institut:	Department of Intelligent Systems (UITS)
Student:	**Frejlach Jakub, Bc.**
Programme:	Information Technology and Artificial Intelligence
Specialization:	Application Development
Title:	**Using Counter-Examples in Controller Synthesis for POMDPs**
Category:	Formal Verification
Academic year:	2022/23

Assignment:

1. Study the state-of-the-art controller synthesis methods for Partially Observable MDPs (POMDPs) with the focus on inductive synthesis methods.
2. Explore if a more general class of counter-examples can be effectively used in the synthesis methods for POMDPs.
3. Extend the existing implementation of the synthesis methods in the tool PAYNT to support a more general class of counter-examples.
4. Using suitable benchmarks, perform a detailed experimental evaluation of the implemented extensions. Focus on the quality of the counter-examples and their generation time in the context of the overall performance of the synthesis process.

Literature:

- Kochenderfer, M.J., Wheeler, T.A., and Wray K.H, Algorithms for Decision Making, MIT Press 2021.
- Andriushchenko, R., Češka, M., Junges, S., and Katoen, J.P. Inductive synthesis of finite-state controllers for POMDPs. In UAI'22. Proceedings of Machine Learning Research.
- Andriushchenko, R., Češka, M., Junges, S., Katoen, J.P. and Stupinský, Š. PAYNT: A Tool for Inductive Synthesis of Probabilistic Programs. In CAV'21. Springer.
- Jantsch, S., Harder, H., Funke, F., and Baier, C. Switss: Computing Small Witnessing Subsystems. In FMCAD'20.

Requirements for the semestral defence:
Items 1, 2, and partially 3.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Češka Milan, doc. RNDr., Ph.D.**
Head of Department:	Hanáček Petr, doc. Dr. Ing.
Beginning of work:	1.11.2022
Submission deadline:	24.5.2023
Approval date:	3.11.2022

Abstract

This thesis examines partially observable Markov decision processes (POMDPs), a prominent stochastic model for decision-making under uncertainty and partial observability. POMDPs have diverse applications, from robot navigation to self-driving vehicles. The undecidable control problem of POMDPs has led to various approaches, including finite-state controllers (FSCs) based on observations and history. Identifying small and verifiable FSCs reduces the synthesis of Markov chains. This thesis focuses on counterexample-guided inductive synthesis (CEGIS) within the PAYNT program, exploring the use of Markov decision processes as counterexamples. A new greedy method for constructing counterexamples is outlined and implemented in PAYNT, showing improvements in some cases compared to the existing method.

Abstrakt

Tato práce se zabývá částečně pozorovatelnými Markovskými rozhodovacími procesy (POMDP), významnými stochastickými modely pro rozhodování za nejistoty a částečné pozorovatelnosti. POMDP lze aplikovat od navigace robotů až po samořídící vozidla. Nerozhodnutelný problém řízení POMDP vedl k různým přístupům, včetně konečných stavových kontrolerů (FSC) založených na pozorování a udržování histore v paměti. Identifikaci malých a ověřitelných FSC lze redukovat na syntézu Markovských řetězců. Tato práce se zaměřuje na induktivní syntézu řízenou protipříklady (CEGIS) implementovanou v rámci programu PAYNT a zkoumá využití Markovských rozhodovacích procesů jako protipříkladů. Je nastíněna nová hladová metoda pro konstrukci protipříkladů, která je implementována v programu PAYNT, která v některých případech vykazuje zlepšení oproti stávající metodě.

Keywords

Markov models, partially observable Markov decision processes, finite-state controler synthesis, probabilistic model checking, counterexample guided inductive synthesis

Klíčová slova

Markovské modely, částečné pozorovatelné Markovské rozhodovací procesy, syntéza konečněstavových kontrolerů, pravděpodobnostní model checking, protipříklady řízená induktivní syntéza

Reference

FREJLACH, Jakub. *Using Counter-Examples in Controller Synthesis for POMDPs.* Brno, 2023. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor doc. RNDr. Milan Češka, Ph.D.

Rozšířený abstrakt

Automatizace byla vždy základním cílem lidstva, od jednoduchých dokola opakujících se úkolů až po složité samohybné roboty a samořídící vozidla schopna samostatně se pohybovat v prostředí a dosáhnout svého cíle. Problém spočívá v navigaci s omezenými nebo kompletně chybějícími informacemi o okolí, což přináší značnou nejistotu. Tato práce se však zaměřuje na jednodušší zařízení jenž ovšem s těmito složitými systémy souvisí.

Nejistotu lze rozdělit do různých typů, přičemž v případě neznámého okolí a nejistých stavů je nejvhodnějším typem stavová nejistota. Tradiční rozhodovací modely, jako jsou Markovské rozhodovací procesy (MDP), postrádají schopnost modelovat stavovou neurčitost, protože jsou plně pozorovatelné. Částečně pozorovatelné Markovské rozhodovací procesy (POMDP) toto omezení řeší začleněním pozorování jako prostředku k získávání informací a vyhodnocování akcí.

Řízení POMDP však představuje nerozhodnutelný problém. Běžně se používají dva základní přístupy. Techniky simulace a posilovaného učení se používají, pokud není k dispozici kompaktní model, ale mohou postrádat interpretovatelnost a zaručenou korektnost. Markovské rozhodovací procesy s věrohodností aktualizují přesvědčení na základě předchozích akcí a nabízejí prostředky pro zvládání stavové neurčitosti, ale trpí škálovatelností a složitými interakcemi. Slibnou alternativu představují konečně stavové kontroléry (FSC), které mapují pozorování na akce na základě získaných informací nebo udržované historie. FSC jsou relativně malé a snadno verifikovatelné.

Identifikace FSC je stejně náročná jako syntéza parametrických Markovských řetězců, což vede k jejich syntéze. Byly navrženy metody jako zjemňování abstrakce (AR), induktivní syntéza řízená protipříkladem (CEGIS) a hybridní přístup. Tato práce se zaměřuje na vylepšení současného přístupu zdokonalením techniky CEGIS. Předchozí výzkum ukázal, že výkonnost CEGIS se liší v závislosti na vstupech, zejména na rozložení programových děr v Markovském řetězci. Práce zkoumá inovativní přístupy ke konstrukci protipříkladů, konkrétně modifikací metody CEGIS na využití protipříkladů založených na Markovském rozhodovacím procesu. Uvažuje se o integraci nástroje A Tool for the Computation of Small WITnessing SubSystems (SWITSS) s PAYNT, ačkoli doba jeho provádění činí výsledky nepoužitelnými. Následně je na základě stávajících technik implementovaných v PAYNT vyvinut hladový algoritmus pro konstrukci protipříkladů založených na Markovském rozhodovacím procesu.

Hlavní přínos této práce je dvojí. Zaprvé, navrhuje metodu pro použití protipříkladů založených na Markovském rozhodovacím procesu v CEGIS metodě, přičemž využívá hladový algoritmus pro konstrukci protipříkladů v rámci programu PAYNT. Metoda využívá jednoduché díry, což jsou parametry rodiny Markovských řetězců nebo konečných stavových kontrolerů v kontextu syntézy POMDP. Jedinečnost těchto děr umožňuje zobecnění a přináší generalizované protipříklady. Implementace obsahuje samostatný modul, který umožňuje přepínat mezi protipříklady Markovského řetězce a Markovského rozhodovacího procesu. Kromě toho práce zkoumá SWITSS, který konstruuje svědecké podsystémy pomocí smíšeného celočíselného lineárního programování, jenž mohou sloužit jako protipříklady. Vzhledem k dlouhým časům konstrukceje však použitelnost těchto protipříkladů omezená.

Za druhé, práce zavádí modifikace hladové metody konstrukce protipříkladů, přičemž experimentuje s randomizací, polohou stavu a apriorními statistikami získanými během syntézy modelu. Základní hladový algoritmus a jeho modifikované verze jsou vyhodnoceny pomocí nejnovějších referenčních modelů částečně pozorovatelných Markovských rozhodovacích procesů z GitHub repozitáře programu PAYNT.

Vylepšením techniky CEGIS a zkoumáním nových přístupů ke konstrukci protipříkladů tato práce přispívá k pokroku v syntéze částečně pozorovatelných Markovových rozhodovacích procesů. Navržené metody a modifikace poskytují poznatky o zlepšení účinnosti a efektivity induktivní syntézy řízené protipříkladem.

Using Counter-Examples in Controller Synthesis for POMDPs

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of doc. RNDr. Milan Češka, Ph.D. Supplementary information was provided by Ing. Roman Andriushchenko. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Jakub Frejlach
May 24, 2023

Acknowledgements

I am deeply grateful to my supervisor, doc. RNDr. Milan Češka, Ph.D., for his invaluable guidance throughout my master's and bachelor's studies. His expertise and support have shaped my academic journey. I also extend my heartfelt thanks to my girlfriend, Veronika, as well as my friends and parents, for their unwavering emotional support.

Contents

List of Figures

Chapter 1

Introduction

One of the eternal goals of mankind has always been automation, the creation of machines capable of carrying out various tasks ranging from simple tasks like repeating just one particular thing repeatedly to highly complex self-moving robots and self-driving vehicles that can navigate throughout the surrounding environment, avoid any obstacles that might be encountered on a way, and reach designated final destination without any human interference. The most significant challenge here is undoubtedly navigation with very little or no information about the surroundings, and therefore uncertainty plays an enormous role. It is worth noting that the purpose of this thesis does not concern such complex devices but rather a much simpler one, whose very own core concept is highly connected to those complex ones.

According to Kochenderfer et. al. [16], there are several types of uncertainty, each of which fits a different scenario. Since in this case the surroundings are unknown and generally the state at which we are currently at is not known as well, apparently the best suit is state uncertainty.

Classic decision making model like Markov decision process (MDP), which modern verification tools like PRISM [10] or STORM [9] are capable to model check, might seem like a good candidate to deal with such conditions, however this model is fully observable and it lacks the ability to model state uncertainty. Such uncertainty can rather be modeled using the so-called partially observable Markov decision processes [22, 15] (POMDP), which define states, actions, similarly to classic Markov decision processes (MDP), and additionally observations, which are means of obtaining information and the results of the actions.

Controlling of POMDPs is, however, an undecidable problem. In general, there are two different fields that deal with the POMDPs control. The first one in the form of simulation [21] and reinforcement learning techniques [13] is usually applied when the model is not available in some compact way, such as models dealing with the concept of infinity, since this approach offers great scalability with the other drawbacks of the problematic interpretation and not really guaranteed correctness. The second option is to employ the formal methods. First one, and perhaps less important for the purpose of this thesis, are belief-state Markov decision processes, where states are a distribution of probability over the states of original POMDP, the so-called beliefs. These beliefs are updated according to previous actions taken [19]. This method similarly to already mentioned reinforcement learning techniques also brings a rather unfriendly ways of interaction together with the size problem of smaller models, for which the beliefs might be huge. The second options is a finite-state controllers (FSCs), which are the alternative representation of history maintained in some inner state

[17]. FCSs possess two undoubtedly beneficial assets, such as a relatively small and compact size and easy verification, which reveal their hidden potential.

Junges et. al. [14] suggests and proves that identifying such an FSC is as difficult as synthesis for parametric Markov chains and thus leads to their synthesis. Andriushchenko et. al. [3] describes current state of the art methods which aims for deterministic FSCs rather than stochastic ones, as those are less difficult to obtain and also benefits from reproducibility of their behavior. FSC is searched for by symbolically representing the design space, which contains finitely many FCSs, as a propositional logic formula and employing learner-teacher framework in which the learner constructs the design space and teacher offers the best possible FSC which learner may either accept or rejects it and adapts the design space. We already mentioned that FSCs searching is possible to reduce to topology synthesis in the Markov chain and thus for the teacher purpose Andriushchenko et. al. [3] proposes the usage of inductive synthesis methods. The key of the aforementioned methods is to investigate the individual members of the Markov chain family and pick the one that satisfies all the constraints. This can be achieved in a variety of ways such as abstraction refinement (AR) which leverages creating a symbolic representation of a family, model checking it and optionally splitting it into subfamilies [8], counterexample-guided inductive synthesis (CEGIS) in which for unsatisfying members the counterexamples are constructed in the form of critical subsystems which then allows removing other members and thus speeding up the exploration process [7], or hybrid solution which combines the best bits of AR and CEGIS [2] where all these three methods are in practice implemented in tool called Probabilistic progrAm sYNThesizer (PAYNT) [4].

The goal of this thesis is to improve the state-of-the-art approach described in [3] by focusing on the CEGIS technique. The key results obtained from article [7] revealed that CEGIS performance fluctuates with different inputs, crucial is the program hole distribution among the Markov chain and thus we aim to find new and innovative approaches to tackle the counterexamples constructing. Specifically we talk about lifting the CEGIS method to use Markov decision process based counterexamples. One way how to achieve this is to use a tool for the computation of Small WITnessing SubSystems (SWITSS) and integrate it to PAYNT to serve as a new way of constructing the counterexamples, it utilizes the reduction of the problem to a mixed-integer linear programming [11, 12]. This however fails miserably as it delivers almost the same results as Markov chain based counterexamples with enormous time overhead. Following these findings we decide to march forward with greedy construction of Markov decision process based counterexamples as the exact solutions and heuristics which SWITSS is able to perform are unusable at this state.

Key contributions

Key contributions of this thesis are:

1. Examination of the usability of the SWITSS program and its potential to provide counterexamples for MDPs by solving mixed-integer linear program.

2. Definition of a greedy method for constructing counterexamples for MDPs based on the greedy method for MC counterexamples implemented in PAYNT tool.

3. Introducing the way how to leverage such greedy method for MDP counterexamples, and MDP counterexamples in general in a PAYNT CEGIS loop.

4. Implementation of the new counterexample generator for MDPs in PAYNT, and contributing such implementation to the PAYNT GitHub repository.

5. Experimental evaluation of the introduced and implemented greedy counterexample for MDP generator on the set latest POMDP models from PAYNT repository.

6. Showing that the method possess a potential as from the results of the evaluation stands out that for several POMDP models this new way of constructing MDP counterexamples surpasses the state of the art results.

Structure of the project

In Chapter 2 we outline the core theory which is considered as a bare theory minimum necessary for this thesis, and that are the Markov chains, Markov decision process, partially observable Markov decision process, and their respective model checking. We will also touch on the key idea of so-called counterexamples with reference to Markov chains and Markov decision process and finite state controllers, which refers to partially observable Markov decision process. Next in Chapter 3 we focus more deeply on inductive synthesis of finite state controllers for partially observable Markov decision processes and its connection with the synthesis of Markov chains from families of Markov chains with parameters. Mainly the counterexample guided inductive synthesis, abstraction refinement synthesis and hybrid dual-oracle synthesis. In the Chapter 4 Python program PAYNT is described and its connection with the synthesis of FSCs and MCs. In Chapter 5 we define counterexamples for Markov decision processes and explore the existing approaches how to construct such counterexamples, mainly the Python program SWITSS and its possible usage in the PAYNT program, integration to the PAYNT program and comparing the results with the current greedy approach. Then we outline the greedy algorithm for Markov decision process counterexamples construction based on the existing algorithm for Markov chain counterexamples. Finally, in Chapter 6 we show a practical approach on how to use Markov decision process based counterexamples in the CEGIS inside the PAYNT program, whose key idea is based on the so-called simple holes (parameters) generalization. We also outline a couple of strategies to attempt a smarter simple holes generalization and at last we experimentally evaluate implemented methods.

Chapter 2

Preliminaries

In this chapter, we introduce an important theory and notation which we use as the basis for this thesis. The preliminary sections will cover some of the core stochastic models and the techniques used for their verification. Firstly, we refer to the Markov chains, one of the simplest models for probabilistic programs. In particular, we are interested in discrete-time Markov chains, a specific type of Markov chain. In this context, we also touch on the principles of model checking and counterexamples, which are both closely related to Markov chain verification. Eventually, the more advanced models referred to as Markov decision processes are discussed, which lifts the Markov chains adding non-determinism. Last but not least, the generalization of Markov decision processes called partially observable Markov decision processes is described. All preliminaries are accompanied by simple and easy-to-understand examples and figures. The Definition 2.1 and Definition 2.2 are taken over and adapted from [7]. The theory for Section 2.1 and its Subsection 2.1.1 about Markov chains and their model checking are from Books [18, 5]. The Subsection 2.1.2 on counterexamples for Markov chains is heavily inspired and adapted from Article [23]. The Section 2.2 and its Subsection 2.2.1 on Markov Decision Processes and its Model Checking are adapted from [8, 10, 1]. Finally Section 2.3 which revolves around Partially observable Markov decision processes is based on Articles [14, 3].

2.1 Markov Chains

The most basic of the Markov models, which is at the same time the core of all the Markov models from which all the other Markov models are derived, is a Markov chain (MC). There are two types of MC, discrete-time Markov chain (DTMC) which works with the discrete points and continuous-time Markov chain (CTMC) operating on whole intervals. Since CTMCs are not the topic of this thesis, we can simplify abbreviations, and thus we will refer to DTMC as MC. The core feature of the MC is a Markov property which states that the next state is determined only by the current state and not the previous states.

Definition 2.1. *A probability distribution over a finite set S is a function $\mu \to [0,1]$ with $\sum_{s \in S} \mu(s) = 1$. Let $Distr(S)$ denote the set of all distributions over S. Let $supp(\mu) = \{s \in S \mid \mu(s) > 0\}$.*

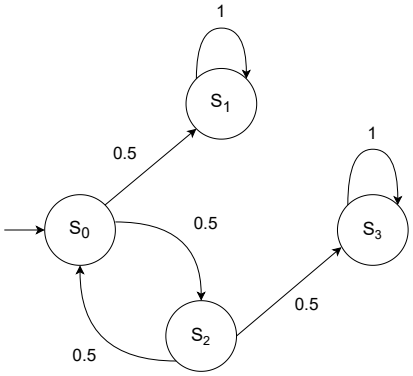
Definition 2.2. *A discrete-time Markov chain (MC) D is defined as a tuple (S, s_0, P) where S is the finite set of states, $s_0 \in S$ is the initial state, and the transition probability matrix $P : S \to Distr(S)$.*

MC is a state-transition system where from each state $s \in S$ the system can make the transition to one of its successor states, this choice is denoted as $P(s)$. This implies that the probability of the system making a transition from state s to t where $s, t \in S$ is denoted as $P(s)(t)$, which could be simplified to $P(s,t)$ for the sake of readability. Then for each state $s \in S$ the set of possible successors is defined as $supp(P(s))$. Iff condition $P(s,s) = 1$ holds, then the state is called the absorbing state because once this state is reached there is no possible transition to any other state.

Path ω is a finite or infinite non-empty sequence of states $s_0, s_1, s_2, ...$ where $\forall i \in N_0 : P(s_i, s_{i+1}) > 0$. This generally holds for any path, so that the state s_0 does not actually have to denote an initial state of MC. The path where $s_0 \in S$ is an actual initial state is called the execution of MC. We denote $Paths^D(s)$ or possibly $Paths^D_{fin}(s)$ to represent a set for all infinite or possibly finite paths that are able to be taken from the state s. Shortcuts $Paths^D$ respectively $Paths^D_{fin}$ denote that the starting point of all the paths in the set is the initial state s_0 of MC D.

$\omega(i)$ denotes an i-th state of the sequence, $|\omega|$ represents the length of the sequence, and for finite paths, we define a $last(\omega)$ to represent a last state of the sequence. Using the Markov property, we can quantify the probability of a specific finite path with the help of a transition probability matrix as follows: $\mathbb{P}[s_0, s_1, s_2, ..., s_n] = \prod_{i=1}^{n-1} P(s_i, s_{i+1})$. However, this approach brings certain difficulties for infinite paths, as the result yields a probability mass zero. This problem can be mitigated by the introduction of cylinder sets. Cylinder set $CS(\omega)$ for a finite path ω is by definition a set of all infinite paths with the common prefix of path ω. Then the probability of $CS(\omega)$ is computed as $P(s_0, s_1) \cdot P(s_1, s_2) \cdot P(s_1, s_2), ...$ [18].

Occasionally, visualization of MC in a graph form is desirable, where nodes represent the states, and edges represent the probabilities that the transition from one state to another will be taken. Transitions with a probability equal to zero are omitted. This is illustrated in Figure 2.1.



(a) MC graph

$$\begin{bmatrix} 0 & 0.5 & 0.5 & 0 \\ 0 & 1 & 0 & 0 \\ 0.5 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(b) MC transition probability matrix

Figure 2.1: Basic MC with 4 states which 2 of them are absorbing.

Example 2.1. *The path $\omega = s_0, s_2, s_0, s_1$ is one of the possible executions of the MC from Figure 2.1. The probability that this execution occurs can be computed as ω is $P(s_0, s_2) \cdot P(s_2, s_0) \cdot P(s_0, s_1) = \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{8}$. In addition, the general probability of reaching the state $s_1 \in S$ eventually can be computed as the sum of the probabilities of all finite paths ending in the state s_1:*

$$\mathbb{P}[s_0, s_1] + \mathbb{P}[s_0, s_2, s_0, s_1] + \mathbb{P}[s_0, s_2, s_0, s_2, s_0, s_1] + ... = \frac{1}{2} + \frac{1}{8} + \frac{1}{32} + ... = \frac{2}{3}$$

or we can take advantage of the cylinder sets, because we can write the set of all the paths that will eventually end up in the state s_1 *as* $\bigcup_{i \in N_0} s_0, (s_2, s_0)^i, s_1$ *and this probability is calculated as:*

$$\sum_{i=0}^{\infty} \mathbb{P}[s_0, (s_2, s_0)^i, s_1] = \frac{1}{2} \cdot \sum_{i=0}^{\infty} \left(\frac{1}{4}\right)^i = \frac{1}{2} \cdot \frac{1}{1 - \frac{1}{4}} = \frac{2}{3}$$

2.1.1 Model Checking MCs

Model checking of MC is a base method to demonstrate the behavior of MC, more precisely the ability to satisfy certain properties. This is typically checked by computing various probabilities throughout the whole model to confirm the likelihood of specific events that either prove or deny the feasibility of those properties. Model checking processes are algorithms that take model specification of an MC together with a set of properties in probabilistic temporal logic, which the model should be verified against, and yield the feasibility results for each property.

Probabilistic Computation Tree Logic (PCTL) is the most common type of temporal logic used for the verification of the MC model.

Definition 2.3. *The syntax of PCTL is as follows:*

$$\Phi ::= true \mid a \mid \neg\Phi \mid \Phi \wedge \Phi \mid P_{\bowtie\lambda}[\phi]$$
$$\phi ::= X\Phi \mid \Phi U^{\leq k} \Phi$$

where a *is an atomic proposition,* $\bowtie \in \{<, \leq, >, \geq\}$, $\lambda \in [0, 1]$ *and* $k \in N \cup \{\infty\}$.

These PCTL formulae are interpreted over the states of MC. There are two types of formulae, state formulae Φ and path formulae ϕ that are evaluated over states or paths, respectively. For the MC properties, only the state formulae are used, since the path formulae serve only as a parameter to a $P_{\bowtie\lambda}[...]$ state formulae operator. MC satisfies the property $P_{\bowtie\lambda}[\phi]$ if the probability of satisfying ϕ lies within the interval specified by $\bowtie \lambda$. As path formulas only $X\Phi$ which denotes that the formula is satisfied in the next state, and $\Phi U^{\leq k}\Psi$ (bounded until) denotes that Φ is satisfied within the k steps and Ψ is true up until that point. Placing the $k = \infty$ we obtain the unbounded until which might be simplified to $\Phi U \Psi$. Diamond operator $\diamond$ (eventually) simplifies the formulas further away, since $\diamond\Phi$ means that Φ is eventually true and $\diamond^{\leq K}\Phi$ analogously that Φ is true within the k steps. For a state s that satisfies the formula Φ or a path ω that satisfies the formula ϕ we write $s \models \Phi$ respectively $\omega \models \phi$ and the negation of the satisfiability $s \not\models \Phi$ or $\omega \not\models \phi$. Properties $\varphi = P_{\bowtie\lambda}[\phi]$ where $\bowtie \in \{<, \leq\}$ are called safety properties and their counterparts where $\bowtie \in \{>, \geq\}$ are called liveness properties.

The core essential property around which this thesis will revolve is the unbounded reachability, as even very complex PCTL formulae may be reduced just to this property. Given the set of target states $T \subseteq S$ property $\varphi \equiv \mathbb{P}[s \models F\ T]$ (we simplify the diamond operator $\diamond$ to F because of readability and compatibility with the information mentioned in

the following chapters) stands for probability that from state $s \in S$ any of the target states in T is eventually reached. The qualitative version of this property $\varphi \equiv \mathbb{P}_{\bowtie\lambda}[s \models F\ T]$ is satisfied iff the probability falls within the given threshold λ and thus $s \models \varphi \Leftrightarrow \mathbb{P}[s \models F\ T] \bowtie \lambda$. Lifting the unbounded reachability to the whole MC, the model D satisfies the property iff the initial state satisfies said property: $D \models \varphi \Leftrightarrow s_0 \models \varphi$.

The algorithmic way of model checking the MC D against the reachability property $\varphi \equiv \mathbb{P}_{\bowtie\lambda}[s \models F\ T]$ consists of obtaining the probability of eventually reaching any of the target states from $T \subseteq S$ for each state $s \in S$ $\mathbb{P}[s \models F\ T]$ and checking the if the $\mathbb{P}[s_0 \models F\ T] \bowtie \lambda$ holds for the initial state of the MC. This computation leads to a solution of a system of linear equations and is shown in detail in Algorithm 1.

Algorithm 1: Computing unbounded reachability probabilities for MC.

Input: MC $D = (S, s_0, P)$, set of target states $T \subseteq S$

Output: Vector with probabilities $x(s) = \mathbb{P}[s \models F\ T]$ for each $s \in S$

1 $S_0 := \{s \in S \mid \mathbb{P}[s \models F\ T] = 0\}$ `/* graph problem */`

2 $S_1 := T$

3 $S_? := S \setminus (S_0 \cup S_1)$

4 Find the solution for the following system of linear equations:

$$x(s) = \begin{cases} 0, & \text{if } s \in S_0 \\ 1, & \text{if } s \in S_1 \\ \sum_{s' \in S} P(s, s') \cdot \boldsymbol{x}(s') & \text{if } s \in S_? \end{cases}$$

5 return x

Example 2.2. *When we return to the MC from Figure 2.1 and Example 2.1 we now can compute the the same probability using this algorithm.* $S_0 = \{s_3\}, S_1 = \{s_1\}$ *and* $S_? = \{s_0, s_2\}$.

$$\begin{aligned} x(s_0) &= \frac{1}{2}x(s_1) + \frac{1}{2}x(s_2) \\ x(s_1) &= 1 \\ x(s_2) &= \frac{1}{2}x(s_0) + \frac{1}{2}x(s_3) \\ x(s_3) &= 0 \end{aligned}$$

Obtaining the vector $x = (\frac{2}{3}, 1, \frac{1}{3}, 0)^T$ *and checking against the result yielded by basic probability or cylinder set computation, we verify that* $\mathbb{P}[F\ \{s_1\}] = \frac{2}{3}$. *With the help of the computed vector of unbounded reachability probabilities of MC* D, *the model can now be verified against various properties, for example* $D \models \mathcal{P}_{>0.5}[F\ \{s_1\}]$, $D \not\models \mathcal{P}_{<0.6}[F\ \{s_1\}]$ *or* $D \not\models \mathcal{P}_{>0.75}[F\ \{s_1\}]$.

Remark. *For the simplicity we focus only on unbounded reachability properties in this thesis, however, there is also a reward-based property defined as* $\varphi \models R_{\bowtie\lambda}[F\ T]$ *denotes a property tied to a expected value of reward in a set of target states* $T \subseteq S$. *More detailed this is described in [20]. Moreover, the reward-based properties will be supported by the algorithms introduced later on.*

2.1.2 Counterexamples for MCs

During the model checking process, if some specification φ is rejected, it is very useful to provide a specific execution of MC that violates the said specification which would act as the so-called counterexample (CE). In general, there are two ways to represent CE in the model-checked MC: either a set of paths or a critical subsystem. Both these approaches are described in great detail in the book [23], therefore we will only mention the essential piece of information.

First, let us dig into the probably more straightforward way, which delivers the CE in the form of a set of paths in MC. Having the specification $\varphi \equiv \mathbb{P}_{\bowtie\lambda}[F\, T]$, the probabilities of such paths must add up so it violates the $\bowtie\ \lambda$ qualitative property. For the safety property $\bowtie \in \{<, \leq\}$ CE is in the form of a set of finite paths that end up in any of the states $s \in T$ whose probabilities summed together exceed λ. In general, we are trying to show that the probability of reaching some state $s \in T$ is higher than the specification value. On the other hand, with the liveness property $\bowtie \in \{>, \geq\}$ it must be shown that the probability of reaching any state $s \in T$ is lower than λ or turning the specification around so that the probability of never reaching any of the states $s \in T$ exceeds $(1 - \lambda)$. This CE can be represented as a set of infinite paths that violate the specification φ and their summed probabilities exceed $(1 - \lambda)$. As already mentioned, it is rather a pitfall to operate on infinite paths, and thus using a cylinder set $\mathrm{CS}(\omega)$ where ω leads to a bottom strongly connected component (BSCC) from where it is impossible to reach any of the states $s \in T$ helps in this case a lot. If the individual probabilities of such SCs accumulate to a higher value than $(1 - \lambda)$ then it can serve as a CE for this specification in the particular MC.

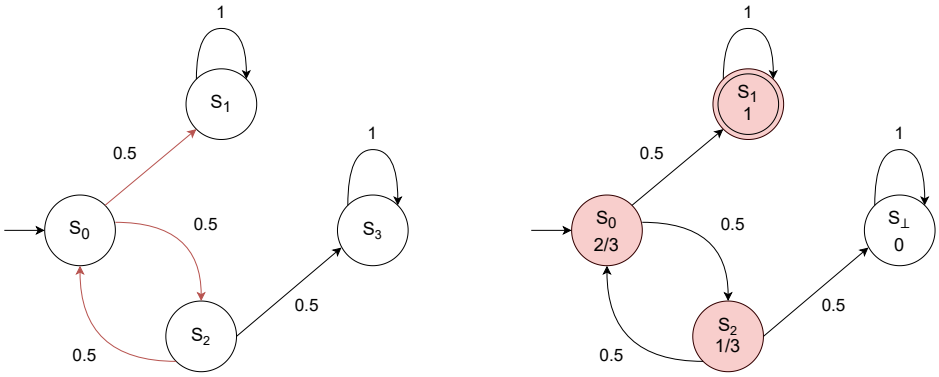
The other way to provide CE for a pair of specification and MC is the critical subsystem. In this thesis, this will be the preferred form of doing so, as this approach is quite compact and handy in contrast with providing a set of paths, since that can sometimes be quite overwhelming, as providing all the possible executions might grow in size quite quickly. Critical subsystem $D{\downarrow}C$ is a fragment of the original MC D that includes only the critical paths and states that already violate the specification φ on their own and hence, even when containing only the portion of the original size of the MC D its sets of paths $Paths^{D\downarrow C}$ and $Paths^{D\downarrow C}_{fin}$ match the CE. Let us formally define this critical subsystem with regard to safety and liveness properties.

Definition 2.4. *Let $D = (S, s_0, P)$ be an MC, state $s_\perp$ such that $s_\perp \notin S$ and $C \subseteq S$ such that $s_0 \in C$. The sub-MC wrt. C is an MC $D{\downarrow}C = (C \cup \{s_\perp\}, s_0, P')$ where the transition probability matrix P' is defined as follows:*

$$P'\left(s, s'\right) = \begin{cases} P\left(s, s'\right) & \text{if } s, s' \in C, \\ 1 - \sum_{s'' \in S \setminus C} P\left(s, s''\right) & \text{if } s \in C \text{ and } s' = s_\perp, \\ 1 & \text{if } s = s' = s_\perp. \end{cases}$$

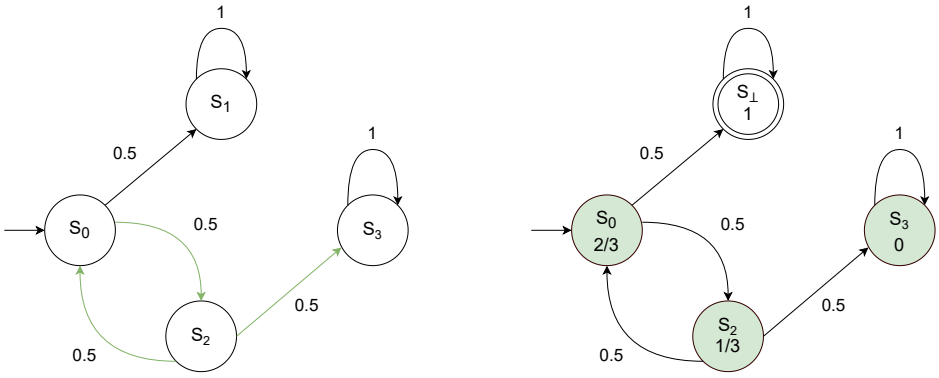
Definition 2.5. *Let $D = (S, s_0, P)$ be an MC, $\varphi \equiv \mathbb{P}_{\leq\lambda}[F\, T]$ be a safety property such that $D \not\models \varphi$ and $\varphi' \equiv \mathbb{P}_{\geq\lambda}[F\, T]$ be a liveness property such that $D \not\models \varphi'$. Then if for some C it holds that $D \downarrow C \not\models \varphi$ as well, then the set C and the corresponding subsystem $D \downarrow C$ are called critical. For liveness property this holds very similarly; however, the corresponding subsystem needs to violate the slightly modified specification such as $D{\downarrow}C \not\models \mathbb{P}_{\geq\lambda}[F\, T \cup \{s_\perp\}]$ because not including this so-called sink state as a valid target state would result in the construction of such tiny CEs that would contain only the initial state of the original MC. When $|C| \leq |C'|$ for every C' then we call it a minimal critical subsystem.*

As already pointed out, critical subsystems will be the form of CE which will serve as a reference point in this thesis. The construction of such a critical subsystem is usually based on gradually expanding the subsystem and adding paths and states from the original MC. The starting point for the expansion is usually the initial state of the MC. After each expansion step, the subsystem is model checked against the specification, and once this specification is not satisfied, the current subsystem may be proclaimed the CE. This algorithm is guaranteed to terminate eventually as the original MC D violates the property for sure and thus the worst case is that the critical subsystem is the whole original MC.



(a) Critical paths (b) Critical subsystem

Figure 2.2: CE for a MC D from Figure 2.1 and safety property $\mathbb{P}_{<0.6}[F\ \{s_1\}]$.



(a) Critical paths (b) Critical subsystem

Figure 2.3: CE for a MC D from Figure 2.1 and liveness property $\mathbb{P}_{>0.75}[F\ \{s_1\}]$.

Example 2.3. *Let us first briefly demonstrate the counterexamples in the form of finite (infinite) paths. Recalling the MC D from Figure 2.1 and the computed unbounded reachability vector in Example 2.2 we can now construct a CE in the form of a set of finite paths* $\{s_0s_1, s_0s_2s_0s_1\}$ *for a specification* $\mathbb{P}_{<0.6}[F\ \{s_1\}]$ *and CE for specification* $\mathbb{P}_{>0.75}[F\ \{s_1\}]$ *as a set of all infinite paths with the common prefix* s_0, s_2, s_3 *as this prefix leads to a BSCC from which the target state may never be reached.*

Now we deliver a critical subsystem corresponding to the CE based on the already shown path. In Figure 2.2a critical paths are colored red. The critical set of states $C = \{s_0, s_1, s_2\}$ *for a safety property* $\mathbb{P}_{<0.6}[F\{s_1\}]$ *induce a critical subsystem* $D{\downarrow}C$ *that violates the property*

$\mathbb{P}_{<0.6}[F\ \{s_1\}]$ *and all states from* $S \setminus C$ *which are not present in the critical subsystem are replaced by the sink state which acts as a common reroute for all transitions that are no longer needed due to the state being replaced by the sink state. A very similar approach may be seen in Figure 2.3a and Figure 2.3b for the liveness property* $\mathbb{P}_{>0.75}[F\ \{s_1\}]$*, but this time the critical paths are colored green and the critical set of states* $C' = \{s_0, s_2, s_3\}$ *induces a different critical subsystem* $D \downarrow C'$ *that violates the modified property* $\mathbb{P}_{>0.75}[F\ \{s_1\} \cup \{s_\perp\}]$*. Adding the sink state* $s_\perp$ *among the target states is a crucial step when constructing the CE for the liveness properties, as leaving out this modification would cause incorrectly constructed CEs that contain only the initial state of the MC.*

2.2 Markov Decision Processes

Markov decision process (MDP) is MC enriched with a new concept of actions which introduce nondeterministic choices to a model. This modification lifts the MC so that now the transitions in the model are driven by both nondeterministic (potentially deterministic as we later introduce the concept of schedulers) choice and the original probability distribution.

Definition 2.6. *A Markov decision process (MDP) is a tuple* $M = (S, s_0, Act, \mathcal{P})$ *where* S *and* s_0 *are identical to those defined in Definition 2.2,* Act *is a finite set of actions and a partial transition probability function* $\mathcal{P} : S \times Act \nrightarrow Distr(S)$*.*

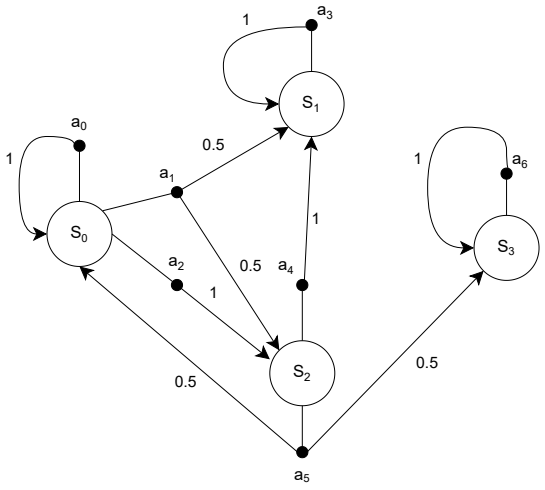
We denote the set of available actions in each state $s \in S$ as $Act(s) = \{a \in Act \mid \mathcal{P}(s, a) \neq \perp\}$. If for all states $s \in S$ in MDP condition $|Act(s)| = 1$ holds, then it is an MC since having only one available action to take from each state removes the nondeterminism and leaves us with only the probability distribution. We employ a similar simplification as we did for the MC and thus $\mathcal{P}(s)(a)(s')$ may be exchanged for $\mathcal{P}(s, a, s')$. The path of an MDP M is either a finite or infinite non-empty sequence consisting of actions and states that we denote like $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots$, where $s_i \in S, a_i \in Act(s_i)$ and $\forall i \in N_0 \mid \mathcal{P}(s_i, a_i, s_{i+1}) > 0$. The set of finite or infinite paths in MDP M is denoted by $Paths^M_{fin}$ or $Paths^M$. Omitting the actions in the path definition induces the (in)finite path definition for MC. For a finite path π we also define the last state $last(\pi)$. The probability of a finite path π could be computed similarly as for a path in MC: $\mathbb{P}[s_0, a_0, s_1, a_1, s_2, a_2, \cdots, s_n] = \prod_{i=1}^{n-1} \mathcal{P}(s_i, a_i, s_{i+1})$.

The same as for MC sometimes the graph visualization is necessary for MDP and a very similar graph form serves such purpose. The nodes represent states and the edges represent transitions with the addition of smaller nodes or possible dots representing possible actions. Outgoing edges from state do not have any probability assigned, and outgoing edges from an action do have the probability assigned. See Figure 2.4 for details.

Since MDP behaves nondeterministically, usage of schedulers is very useful for dealing with nondeterminism.

Definition 2.7. *A scheduler for an MDP* $M = (S, s_0, Act, \mathcal{P})$ *is a function* $\sigma : Paths^M_{fin} \to Act$ *such that* $\sigma(\pi) \in Act(last(\pi))$ *for all* π *in* $Paths^M_{fin}$*. Scheduler* σ *is memoryless if* $last(\pi) = last(\pi') \Rightarrow \sigma(\pi) = \sigma(\pi')$ *for all* $\pi, \pi' \in Paths^M_{fin}$*. The set of all MDP* M *schedulers is* Σ^M*.*

Definition 2.8. *The MC induced by MDP* M *and the scheduler* $\sigma \in \Sigma^M$ *is given by* $M^\sigma = (Paths^M_{fin}, s_0, P^\sigma)$ *where:*



(a) MDP graph

$$\begin{array}{c} \\ (s_0,a_0) \\ (s_0,a_1) \\ (s_0,a_2) \\ (s_1,a_3) \\ (s_2,a_4) \\ (s_2,a_5) \\ (s_3,a_6) \end{array} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.5 & 0.5 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0.5 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(b) MDP transition probability matrix for individual actions.

Figure 2.4: Basic MDP with 4 states and 7 actions.

$$P^{\sigma}\left(\pi,\pi'\right) = \begin{cases} \mathcal{P}(last(\pi),\sigma(\pi),s') & \text{if } \pi' = \pi \xrightarrow{\sigma(\pi)} s' \\ 0 & \text{otherwise.} \end{cases}$$

Scheduler basically drives deterministically choices in each MDP state. Whenever the state $last(\pi)$ of the path π is reached, the scheduler takes action $a = \sigma(\pi) \in Act(last(\pi))$ and then the transition is made based on the probability distribution $\mathcal{P}(last(\pi), a)$. Specifying this further away, if the scheduler σ is memoryless, mapping from state to actions is obtained as no matter the path the scheduler will always choose action $\sigma(s)$ in state $s \in S$.

Example 2.4. *Let us use MDP* $M = (S, s_0, Act, \mathcal{P})$ *where* $S = \{s_0, s_1, s_2, s_3\}$, $Act = \{a_0, a_1, a_2, a_3, a_4, a_5, a_6\}$ *and* $\mathcal{P}$ *defined in Figure 2.4 by both graph and transition probability matrix for individual actions. Path* $\pi = s_0 \xrightarrow{a_2} s_2 \xrightarrow{a_5} s_0 \xrightarrow{a_0} s_0 \xrightarrow{a_1} s_1$ *is one of the possible executions of MC* M*. The probability of such a path is computed as:*

$$\mathbb{P}[\pi] = \mathcal{P}(s_0, a_2, s_2) \cdot \mathcal{P}(s_2, a_5, s_0) \cdot \mathcal{P}(s_0, a_0, s_1) = 1 \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$$

We can induce MC M^{σ} *from MDP* M *applying the memoryless scheduler* $\sigma \in \Sigma^M$ *where* $\sigma : [s_0 \mapsto a_1, s_1 \mapsto a_3, s_2 \mapsto a_5, s_3 \mapsto a_6]$ *which is equivalent to MC* D *from Figure 2.1.*

2.2.1 Model Checking MDPs

MDP model checking uses the same specification language as MC model checking; however, there are certain differences between those two processes and thus also in the semantics of the language. The main change resides in the fact that unlike in the MC model checking, where only one possible outcome was possible, we now might have a set of action in each of the MDP states to choose from, and thus the resulting probability of reaching some state from another clearly depends on which actions are favored. We say that the specification φ holds for an MDP M iff it holds for the induced MCs of all schedulers, that is, $M \models \varphi \Leftrightarrow \forall \sigma \in \Sigma^{\sigma} : M^{\sigma} \models \varphi$. Since the number of schedulers for any MDP is infinite, we restrict the search to only memoryless ones and hence refine the model checking to simple

finding of maximum or minimum probability with the corresponding scheduler as shown in Proposition 2.1.

Proposition 2.1. *Let $M = (S, s_0, Act, \mathcal{P})$ and φ be a property. Then $\sigma_{min}, \sigma_{max} \in \Sigma^M$ denotes the memoryless schedulers (minimizing/maximizing) such that $\forall \sigma \in \Sigma^M : \mathbb{P}[M^{\sigma_{min}} \models \varphi] \leq \mathbb{P}[M^{\sigma} \models \varphi] \leq \mathbb{P}[M^{\sigma_{max}} \models \varphi]$.*

For a particular specification $\varphi \equiv \mathbb{P}_{\bowtie\lambda}[F\,T]$ we need to compute a minimizing scheduler σ_{min} in case of liveness property or possibly maximizing scheduler σ_{max} in safety property scenario. Obtaining such a scheduler allows us to compute the minimum / maximum probability of reaching T from each state that is $\forall s \in S : x(s) := aggr_{\sigma \in \Sigma^M} \mathbb{P}[M^{\sigma}, s \models F\,T]$ where $aggr \in \{min, max\}$ and then simply lay down and assert $x_{min}(s_0) \geq \lambda$ for liveness or $x_{max}(s_0) \leq \lambda$ for safety. This computation of the lower (x_{min}) and upper (x_{max}) bounds of reachability probabilities leads to a solution of mixed-integer linear program (MILP). For a simplification purposes we also may refer to $x_{min}(s_0)$ and $x_{max}(s_0)$ as $\mathbb{P}_{min}$ and $\mathbb{P}_{max}$. In Algorithm 2 a detailed approach is described to compute x_{max}. By rearranging some of the operators, we can obtain the steps for computing the x_{min} as well. MILP solution gives the exact answers; however, there is a significant drawback with scalability on larger models and thus value iteration or policy iteration methods are usually applied instead of MILP solving [10].

Algorithm 2: Computing unbounded reachability probabilities for MDP.

Input: MDP $M = (S, s_0, Act, \mathcal{P})$, set of target states $T \subseteq S$

Output: Vector with probabilities $x_{max}(s) = max_{\sigma \in \Sigma^M} \mathbb{P}[M^{\sigma} s \models F\,T]$ for each $s \in S$

1 $S_0 := \{s \in S | \forall \sigma \in \Sigma^M : \mathbb{P}[M^{\sigma}, s \models \varphi] = 0\}$ `/* graph problem */`

2 $S_1 := \{s \in S | \exists \sigma \in \Sigma^M : \mathbb{P}[M^{\sigma}, s \models \varphi] = 1\}$ `/* graph problem */`

3 $S_? := S \setminus (S_0 \cup S_1)$

4 Find x_{max} as solution to MILP where maximizing the $\sum_{s \in S} x(s)$ subject to

$$\forall s \in S_0 : x(s) = 0$$
$$\forall s \in S_1 : x(s) = 1$$
$$\forall s \in S_? \forall a \in Act(s) : x(s) \geq \sum_{s' \in S} \mathcal{P}\left(s, a, s'\right) \cdot x\left(s'\right)$$

5 return x_{max}

2.3 Partially Observable Markov Decision Processes

Partially observable Markov decision process (POMDP) is a generalization of MDP where we deal with state uncertainty. The agent usually has incomplete information to work with. We call this piece of information an observation. Agent can then no longer observe the underlying state of the model and needs to do decisions based on such observations compared to already introduced models like MC or MDP.

Definition 2.9. *A partially observable MDP (POMDP) is a tuple $\mathcal{M} = (M, Z, O)$ where $M = (S, s_0, Act, \mathcal{P})$ is a underlying MDP of $\mathcal{M}$, Z is a finite set of observations and a*

(deterministic) observation function $O : S \to Z$. *The observation* Z *is called trivial or perfect if there is only one state* $s \in S$ *with* $O(s) = z$ *for some observation* $z \in Z$.

Lifting the observation function O to paths, such that for $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots s_n \in Paths^M_{fin}$ we get the observation sequence $O(\pi) = O(s_0) \xrightarrow{a_0} O(s_1) \xrightarrow{a_1} \cdots O(s_n)$.

Definition 2.10. *An observation-based strategy* ρ *for a POMDP* $\mathcal{M}$ *is a strategy for underlying MDP* M *such that* $\forall \omega, \omega' \in Paths^M_{fin} \mid \rho(\omega) = \rho(\omega')$ *with* $O(\omega) = O(\omega')$. $\sum^{\mathcal{M}}$ *is the set of observation-based strategies for POMDP* $\mathcal{M}$.

Observation-based strategy actions based on observations along a path and the past actions. Induced MC M^ρ is obtained be applying some observation-based strategy ρ on POMDP. For compact representation of such strategies with finite memory finite-state controllers (FSCs) are defined. There is a variety of possible FSC representation, but we will restrict our usage to Mealy machines with the output determined by taken transition and also we restrict only to deterministic FSCs.

Definition 2.11. *A (deterministic) finite-state controller (FSC) for a POMDP* $\mathcal{M}$ *is a tuple* $\mathcal{F} = (N, n_0, \gamma, \delta)$, *where* N *is a finite set of memory nodes,* $n_0 \in N$ *is the initial memory node,* γ *is the action mapping* $\gamma : N \times Z \to Act$ *which determines the action when the agent is in node* n *and observers* z *and* δ *is the memory update* $\delta : N \times Z \to N$ *which updates to a new node based on presence in a particular node* n *and observing* z. *For* $|N| = k$ *we call an FSC a* k*-FSC. Let* $\rho_{\mathcal{F}} \in \Sigma^{\mathcal{M}}$ *denote the observation-based strategy represented by* $\mathcal{F}$.

Similarly as observation-based strategy, k-FSC $\mathcal{F}$ may also be applied to POMDP $\mathcal{M}$ to obtain induced MC $M^{\mathcal{F}} = (S^F, (s_0, n_0), P^F)$ where $S^F = S \times N$ and with the usage of $z = O(s)$:

$$P^F((s,n),(s',n')) = P(s,s',\gamma(n,z)) \cdot [n = \delta(n,z)]$$

where Iverson-brackets implies that $[x] = 1$ if predicate x is true and 0 otherwise.

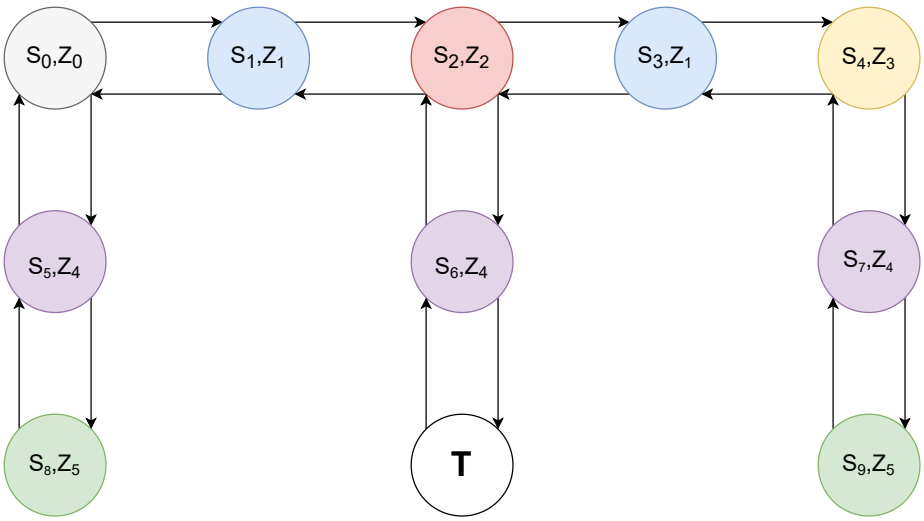
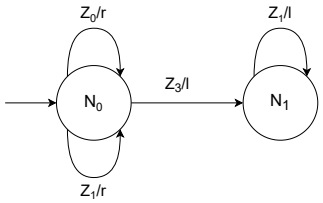
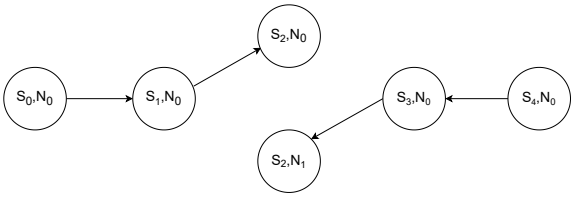


Figure 2.5: A simple maze problem represented as POMDP (adapted from [3]).



(a) Part of a 2-FSC (adapted from [3]).



(b) Part of the induced MC (adapted from [3]).

Figure 2.6: Part of a FSC and corresponding induced MC for POMDP from Figure 2.5.

Example 2.5. *In Figure 2.5 maze problem is depicted as a POMDP* $\mathcal{M}$ *with* $S = \{s_0, s_1, \cdots, s_T\}$, $Act = u, d, r, l$ *and* $Z = \{z_0, \cdots, z_5\}$. *In Figure 2.6a a fragment in form of 2-FSC may be seen with* $\gamma(n_0, z_0) = \gamma(n_0, z_1) = r$ *(observing* z_0 *or* z_1 *in memory node* n_0 *results in action* r*) and* $\gamma(n_0, z_3) = \gamma(n_1, z_1) = l$ *(observing* z_3 *in memory node* n_0 *or observing* z_1 *in memory node* n_1 *yield action* l*). Finally Figure 2.6b depicts induced MC obtained by applying the 2-FSC onto POMDP* $\mathcal{M}$.

Chapter 3

Inductive Synthesis Methods for POMDPs

From Section 2.3 a certain connection between an FSC and an MC is indicated. This is a key part of the information on which the idea of controller synthesis is based, which is one of the possible approaches to tackle the POMDP state uncertainty, which will be described in more detail. With the usage of parameters, families of MCs can represent many different realizations with changing topologies and an immutable set of states [8]. Leveraging the families, one can ask the question, how to synthesize an MC from the family of MCs to satisfy certain specifications? Similarly, based on the connection between the FSC and an MC, which will be explained in the following section, the whole concept of MC families with parameters may be lifted and applied to FSCs as well. Various synthesis methods such as counterexample-guided inductive synthesis [7], abstraction refinement [8], or the dual hybrid method [1, 2] can be employed to synthesize searched MC, respectively, FSC.

3.1 Families of FSCs

From what we've learned in Section 2.3 a POMDP and a single FSC result into a single induced MC and hence a POMDP and a set of FSCs induces a set of MCs. In addition, FSC sets have an additional structure that allows for a concise description of MC sets.

Definition 3.1. *A family of full k-FSCs is a tuple* $\mathcal{F}_k = (N, _0, K)$*, where* N *is a set consisting of* k *nodes,* $n_0 \in N$ *is the initial node and* $K = N \times Z$ *is a finite set of parameters where each has its own domain* $V_{(n,z)} \subseteq Act \times N$*.*

By fixing the value of each parameter, one may obtain a k-FSC from a family which shows that each family describes a set of FSCs by varying in the substitutions of the parameters. Such families are usually described as $\mathcal{F}_k$. Generally speaking, a POMDP $\mathcal{M}$ and a family $\mathcal{F}_k$ induces the family of MCs $\mathcal{M}^{F_k} = \{M^F | F \in \mathcal{F}_k\}$ [3].

Remark. *From now on we focus families of MCs and ways of their synthesis. As we already showed the connection between the family of MCs and FSCs, the following synthesis algorithms will be understood more clearly in the MC world and hence it is just important to remember that all the following synthesis methods might be used for FSC synthesis as well.*

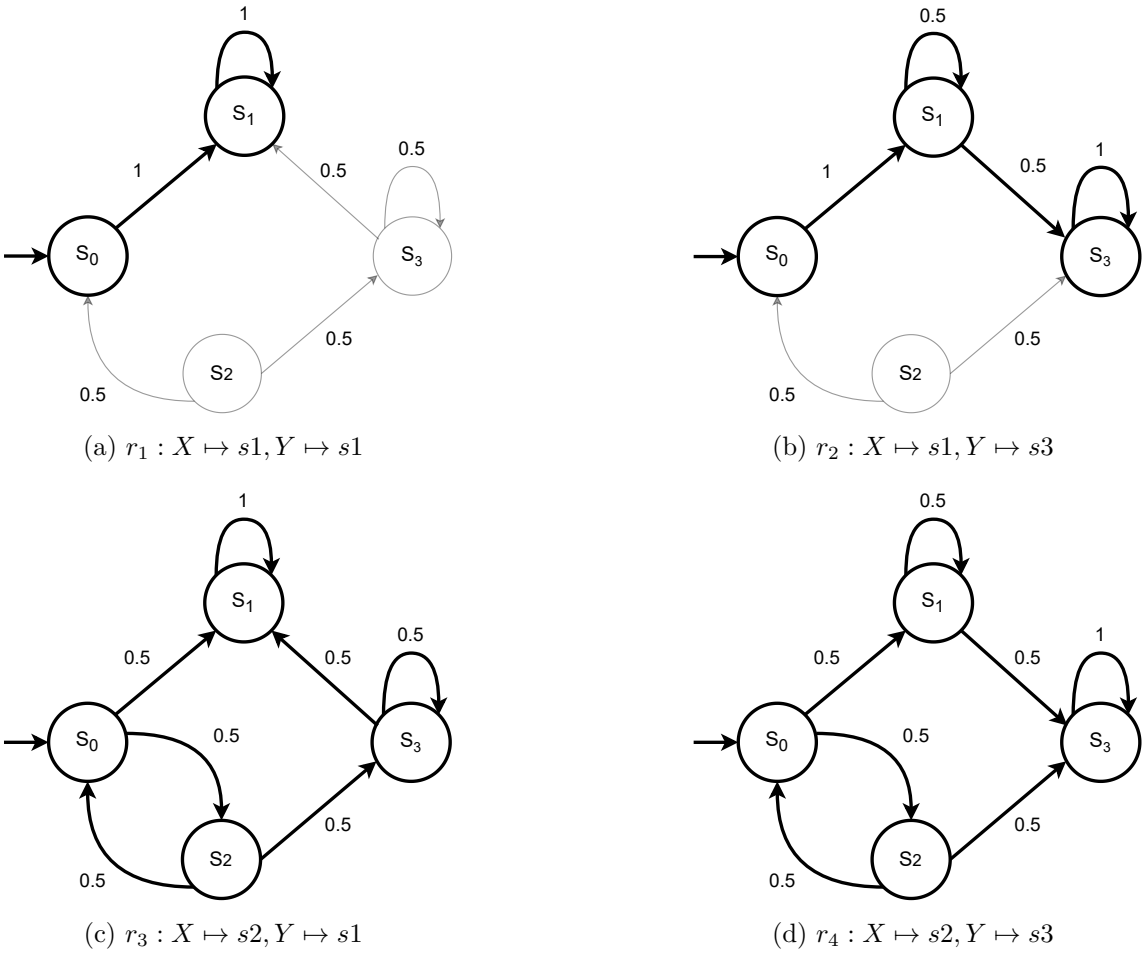
3.2 Families of MCs

Definition 3.2. *A family of MCs is a tuple $\mathcal{D} = (S, s_0, K, \mathcal{B})$ where S and s_0 are defined the same as in Definition 2.2, K is a finite set of parameters with domains $T_k \subseteq S$ for each parameter $k \in K$, and $\mathcal{B} : S \to Distr(K)$ is a family of transition probability functions.*

The function $\mathcal{B}$ maps each state $s \in S$ to a distribution over parameters K. In contrast to MC synthesis, such parameters represent unknown options (holes) of a specific model. The assignment of a specific value, more precisely state, to each of the parameters yields MC, which represents a concrete realization of a family $\mathcal{D}$, which is described in the following definition.

Definition 3.3. *A realisation of a family $\mathcal{D} = (S, s_0, K, \mathcal{B})$ of MCs is a function $r : K \to S$ such that $\forall k \in K : r(k) \in T_k$. Realisation r induces MC $\mathcal{D}_r = (S, s_0, \mathcal{B}_r)$ iff $\mathcal{B}_r(s, s') = \sum_{k \in K, r(k)=s'} \mathcal{B}(s)(k)$ for all pairs of states $s, s' \in S$. Let $\mathcal{R}^{\mathcal{D}} = \prod_{k \in K} T_k$ denote the set of all the realizations of $\mathcal{D}$.*

The family $\mathcal{D}$ has finite parameter domains and thus the number of family realisations is finite too, however, exponential in $|K|$. All MCs from one family $\mathcal{D}$ share the same immutable state space and their topology, such that reachable states may be different.



(a) $r_1 : X \mapsto s1, Y \mapsto s1$

(b) $r_2 : X \mapsto s1, Y \mapsto s3$

(c) $r_3 : X \mapsto s2, Y \mapsto s1$

(d) $r_4 : X \mapsto s2, Y \mapsto s3$

Figure 3.1: Family of 4 MCs $\mathcal{D}$. Grayed out parts of the MC are unreachable.

Example 3.1. *Let us assume a family from Figure 3.1* $\mathcal{D} = (S, s_0, K, \mathcal{B})$ *of MCs with the state space* $S = \{s_0, s_1, s_2, s_3\}$ *and a set of parameters* $K = \{X, Y, k_0, k_1, k_3\}$ *and their respective domains* $T_X = \{s_1, s_2\}$, $T_Y = \{s_1, s_3\}$, $T_{k_0} = \{s_0\}$, $T_{k_1} = \{s_1\}$, $T_{k_3} = \{s_3\}$ *(notice that parameters* k_0, k_1, k_3 *have domains of size 1 and hence are not considered to be parameters for the simplification) and the family of transition probability functions* $\mathcal{B}$ *defined in a PRISM style (will be explained later in Chapter 4):*

$$\begin{aligned}
\mathcal{B}(s_0) &= \frac{1}{2} : k_1 + \frac{1}{2} : X, \\
\mathcal{B}(s_1) &= \frac{1}{2} : k_1 + \frac{1}{2} : Y, \\
\mathcal{B}(s_2) &= \frac{1}{2} : k_0 + \frac{1}{2} : k_3, \\
\mathcal{B}(s_3) &= \frac{1}{2} : k_3 + \frac{1}{2} : Y.
\end{aligned}$$

Currently, there are two defined synthesis problems that the following synthesis methods are able to tackle. When both the state space S and the set of parameters K are finite, then both of these problems are decidable, more precisely, $\mathcal{NP}$-hard. Let us assume some specification $\varphi \equiv \mathbb{P}_{\bowtie\lambda}[F\ T]$.

1. **Feasability Synthesis**: For a family of MCs $\mathcal{D}$ and the specification φ identify a realisation $r \in \mathcal{R}^{\mathcal{D}}$ such that for an induced MC $\mathcal{D}_r \models \varphi$.
2. **Maximum Synthesis (optimality)**: For a family of MCs $\mathcal{D}$ and specification φ identify a realisation $r^* \in \mathcal{R}^{\mathcal{D}}$ such that $r^* \in argmax_{r \in \mathcal{R}^{\mathcal{D}}} \mathbb{P}[\mathcal{D}_r \models F\ T]$.

The feasability synthesis problem is basically searching for realisation satisfying all the given specifications, which may also result in finding no such realisation. The problem of maximum synthesis then describes finding a realisation that maximizes the probability of reaching a set of target states T. Minimizing is defined analogously.

A very trivial approach may be used to solve the synthesis problem, which is the so-called one-by-one method. In other words, brute forcing through each realisation $r \in \mathcal{R}^{\mathcal{D}}$ until either satisfying realisation is found or the specification is proclaimed unfeasible. However, this method only serves as a proof of concept since for larger families this is unacceptable due to a state space and parameter space explosion. More sophisticated methods will be described in the following sections. These methods take advantage of inspecting whole subfamilies (sets of realisations) or generalizing the results of analysis to subfamilies [1, 4].

Definition 3.4. *Let* $\mathcal{D} = (S, s_0, K, \mathcal{B})$ *be a family of MCs, and* $\mathcal{R} \subseteq \mathcal{R}^{\mathcal{D}}$ *a subset of realisations. A subfamily of* $\mathcal{D}$ *consisting of realisations from* $\mathcal{R}$ *is a family* $\mathcal{D}[\mathcal{R}] = (S, s_0, K, \mathcal{B})$ *where the set of all realisations of the subfamily* $\mathcal{R}^{\mathcal{D}[\mathcal{R}]} = \mathcal{R}$.

Definition 3.5. *Let* $\mathcal{D} = (S, s_0, K, \mathcal{B})$ *be a family of MCs and* $r \in \mathcal{R}^{\mathcal{D}}$ *be any realisation. A generalization of* r *in favor of the subset* $\overline{K} \subseteq K$ *of parameters is a set* $r \uparrow \overline{K} = \{r' \in \mathcal{R}^{\mathcal{D}} | \forall k \in \overline{K} : r(k) = r'(k)\}$.

In other words, a generalization set $r \uparrow \overline{K}$ defines a maximum set of all realizations that share the same assignment of parameters from the set $\overline{K}$. Other parameters are not important.

3.3 Counterexample-Guided Inductive Synthesis

First possible advanced synthesis method which we will describe and which also a main contributions of this thesis will revolve around is a counterexample-guided inductive synthesis (CEGIS) introduced in [7]. Core ideas of this synthesis approach are built on top of the one-by-one approach with possible pruning some of the realisations and thus reducing the space to explore, it goes as follows. Assuming the set of all realisations of a family of MCs $\mathcal{R}^{\mathcal{D}}$, we randomly pick some realization $r \in \mathcal{R}^{\mathcal{D}}$ and construct the corresponding induced MC $\mathcal{D}_r$. Let φ be a property to examine. We model check whether $\mathcal{D}_r \models \varphi$, if the property is satisfied, a satisfying solution (assignment of parameters) is returned in the form of the realisation r. If the property, on the other hand, is not met, the critical set of states C for induced MC $\mathcal{D}_r$ and property φ is computed. Since the critical set C usually contains only a fragment of the original states S, the subsystem $\mathcal{D}_r{\downarrow}C$, which serves as a CE, can then be constructed with the omission of some of the parameters $k \in K$, namely those whose assignment is not relevant.

Definition 3.6. *Let $\mathcal{D} = (S, s_0, K, \mathcal{B})$ be a family of MCs. For a critical subset of states $C \subseteq S$, a set of relevant parameters (conflict) is obtained as $\overline{K} = \cup_{s \in C} supp(\mathcal{B}(s))$.*

Using the set of relevant parameters obtained, we have a generalization $r \uparrow \overline{K}$. Since any realisation from the generalization set $r \uparrow \overline{K}$ may differ from r only in irrelevant parameters, it must hold that none of these realisation satisfies the property φ, same as realisation r. Formally, we write $\mathcal{D}_r \not\models \varphi \Rightarrow \forall r' \in r \uparrow \overline{K} : \mathcal{D}_{r'} \not\models \varphi$. This way all realisations from the subfamily generalization are pruned in once, and they don't have to be model checked separately. By rejecting the subfamily $\mathcal{D}[r \uparrow \overline{K}]$ the generalization set $r \uparrow \overline{K}$ is subtracted from the set of all realisations $\mathcal{R}^{\mathcal{D}}$. This approach is repeated in a loop and will result in either finding some realization which creates accepting assignment or proving that feasible assignment does not exist for this pair of particular family of MCs and property. The smaller the conflict $C \subseteq S$, the larger the size of the pruned space is. In Algorithm 3 and also in Figure 3.2 the CEGIS synthesis approach is described in more depth.

Algorithm 3: Counterexample-guided inductive synthesis.

Input: A family of MCs $\mathcal{D} = (S, s_0, K, \mathcal{B})$, property φ
Output: Realisation $r \in \mathcal{R}^{\mathcal{D}}$ such that $\mathcal{D}_r \models \varphi$ or UNSAT in the case of non-existing such realisation.

```
1 CEGIS (D, φ):
2     while R^D ≠ ∅ do
3         r := pickRealisation(R^D)
4         if D_r ⊨ φ then
5             return r
6         D_r↓C := constructCriticalSubsystem(D_r, φ)
7         K̄ := identifyRelevantParameters(D, D_r↓C)
8         R^D := R^D \ r ↑ K̄
9     return UNSAT
```

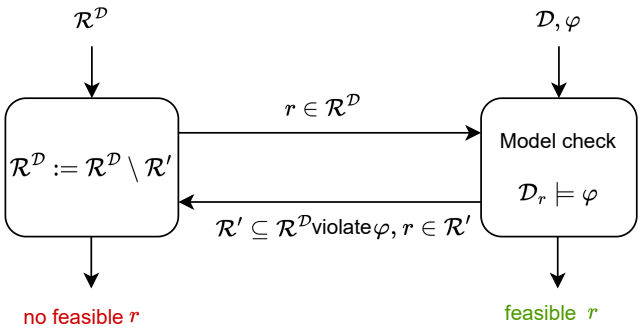


Figure 3.2: Counterexmaple-guided inductive synthesis (adapted from [4]).

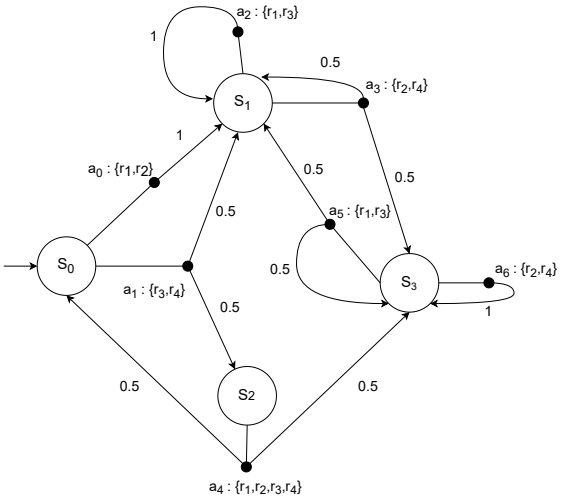
3.4 Abstraction Refinement

The second synthesis method, which we will mention briefly, is called abstraction refinement (AR) [8]. AR approach is completely opposite from the CEGIS, as unlike CEGIS, instead of the one-by-one analysis of the all the realisations of a family of MCs accompanied with the possible pruning of some realisations having the same set of critical states, it focuses on examining whole sets of realisations. To analyze a whole set of realisations at once, a special stochastic process with all the realisations from the examined set, enabled at the same time. More precisely, from each state $s \in S$ of the mentioned process, there is a nondeterministic choice of realisation $r \in \mathcal{R}^{\mathcal{D}}$ which simulates the parameters from the assignment of the set K. This stochastic model leads to an MDP representation, and informally it is called a quotient MDP. We outline only the essential information about the MDP quotient in Definition 3.7, for a more detailed description on how to obtain the MDP quotient, see article [8].

Definition 3.7. *Let $\mathcal{D} = (S, s_0, K, \mathcal{B})$ be a family of MCs. A quotient MDP of family $\mathcal{D}$ is a MDP $M^{\mathcal{D}} = (S, s_0, \mathcal{R}^{\mathcal{D}}, \mathcal{P})$ where $\mathcal{P}(\cdot)(r) \equiv \mathcal{B}_r$. For the subset $\mathcal{R} \subseteq \mathcal{R}^{\mathcal{D}}$, a restriction of $M^{\mathcal{D}}$ with reference to $\mathcal{R}$ is an MDP $M^{\mathcal{D}}[\mathcal{R}]$, which might be simplified to $M^{\mathcal{D}[\mathcal{R}]}$.*

Such a quotient MDP is an over-approximating abstraction of some family of MCs as it over-approximates its each realisation. This furthermore means that the quotient MDP is capable of simulating any realisation from the given family and it can even switch currently executed realisations on the fly during the run-time. Since this is an over-approximation of a family, we need to keep in mind that it is then possible to identify invalid paths in resulting quotient MDP, that or not consistent, because they are simply combination of multiple realisations and they do not exist solely within just one realisation. The example quotient MDP and its transition probability matrix of individual actions, or more precisely realisations, is depicted in Figure 3.3.

Now we describe the actual synthesis method. From the input set of realisations $\mathcal{R}^{\mathcal{D}}$ the quotient MDP is constructed. Let φ again be the property to analyze. In the model checking phase, the minimizing and maximizing bounds σ_{min} and σ_{max} along with the corresponding lower and upper bounds x_{min} and x_{max} are computed and based on the bounds the set of realisation is is either accepted, rejected or refined. Now let us assume that property φ is a reachability property with threshold λ, if $x_{max} \leq \lambda$ for a safety property or $x_{min} \geq \lambda$ for a liveness property then it is safe to assume that each realisation $r \in \mathcal{R}$ satisfies φ. Analogously, if $x_{max} \leq \lambda$ for a safety property or $x_{min} \geq \lambda$ for a liveness property, then



(a) Quotient MDP graph.

$$\begin{array}{c} (s_0,a_0) \\ (s_0,a_1) \\ (s_1,a_2) \\ (s_1,a_3) \\ (s_2,a_4) \\ (s_3,a_5) \\ (s_3,a_6) \end{array} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0.5 & 0.5 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0.5 & 0 & 0.5 \\ 0.5 & 0 & 0.5 & 0 \\ 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(b) Quotient MDP transition probability matrix for individual actions (realisations).

Figure 3.3: Quotient MDP for family of MCs depicted in Figure 3.1.

each realisation $r \in \mathcal{R}^{\mathcal{D}}$ does not satisfy property φ and hence no feasible solution is found. The third case occurs when $x_{min} \leq \lambda \leq x_{max}$ and in such a case nothing can be concluded and the current family needs to be split into two subfamilies $\mathcal{R}_{\top}$ and $\mathcal{R}_{\bot}$ and these two subfamilies are then analyzed separately using the AR method and this is repeated until a feasible solution is found or there is no feasible solution. Schema in Figure 3.4 describes AR in more detail.

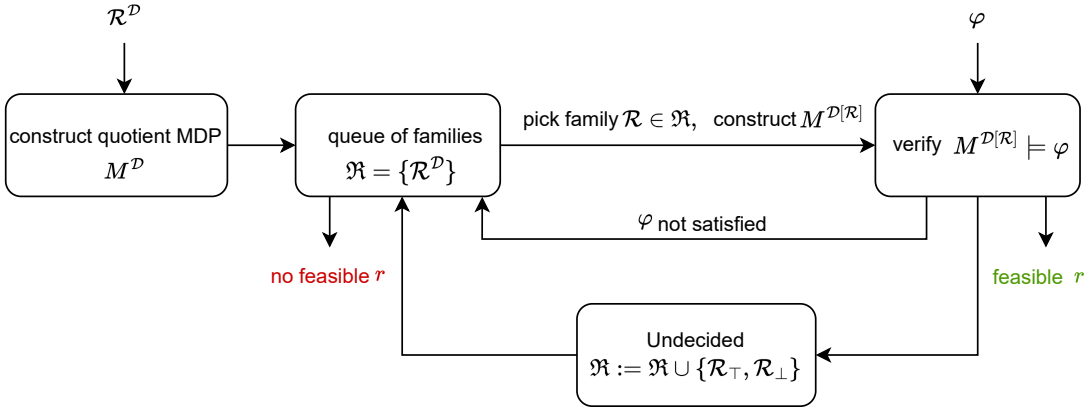


Figure 3.4: Abstraction refinement synthesis.

3.5 Hybrid Dual-Oracle Synthesis

Both the previously mentioned synthesis methods, CEGIS and AR, have their strengths and weaknesses. Both are able to perform really well, but their efficiency depends on the topology. It is very common that for one family of MCs CEGIS is able to perform really well and AR lacks behind and the efficiency swaps for some other topology. This unstable performance leads to a combination of both techniques called hybrid dual-oracle synthesis

(abbreviated simply hybrid). Hybrid uses best bits of both CEGIS and AR in such a way that it alternates between the two methods and allocates time to the synthesis method, which performs better at the moment. It was first introduced in [1] and was improved in [4], as it allowed to use the bounds obtained from AR in CEGIS to get smaller CEs. The graphical schema of the functionality of the hybrid synthesis method is shown in Figure 3.5.

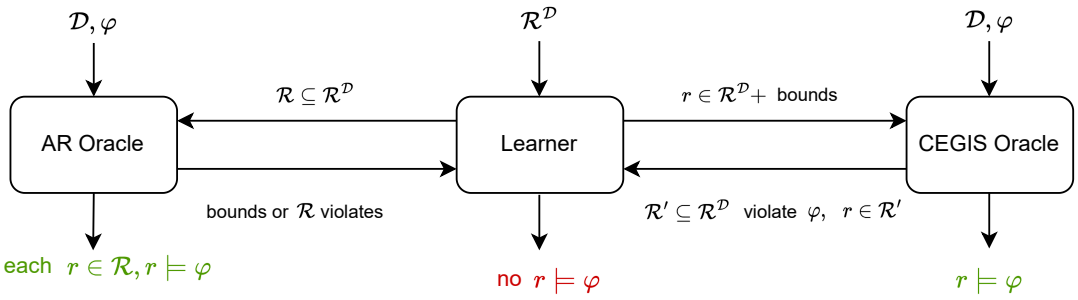


Figure 3.5: Hybrid Dual-Oracle synthesis (adapted from [4]).

Chapter 4

PAYNT - Probabilistic prograAm sYNThesizer

Previous chapter outlined the connection between MC synthesis and FSC synthesis and described three core methods used for the purpose of synthesis. These methods are implemented in the Probabilistic prograAm sYNThesizer (PAYNT)[1] tool, written in Python programming language, first introduced in article [4], which is built on top of the STORM model checker[2] written in C++ language [9]. Stormpy Python bindings[3] are leveraged as middleware between Python PAYNT and C++ Storm. The original purpose of this program was to perform synthesis of probabilistic programs but is now extended with the ability to synthesize FSCs as well. This chapter will further describe the connection between the synthesis of MCs/FSCs and probabilistic programs and how PAYNT is adjusted for the FSC synthesis.

4.1 PAYNT for probabilistic programs

Despite the fact that MCs are models designed to describe probabilistic systems, in the real world, they are no longer so practical due to a state space explosion. Instead of pure MCs, such probabilistic systems are usually described with high-level programming languages that are designed exactly for such purposes. The talk is about languages like PRISM [10], JANI [6], and other languages for similar purposes. Only from this description is the actual MC constructed.

With reference to MC families with parameters described in Section 3.2 the so-called sketches [7] that represent the incomplete high-level program which contains undefined parameters with their respective domains from which they need to be assigned to induce MC. These undefined parameters in the sketch are called holes. Therefore, the probabilistic synthesis is then responsible for finding the assignment solution for these holes with the STORM model checker assistance.

4.1.1 PRISM Sketch Language

Program written in PRISM language consists of one or more reactive modules. These modules are able to interact with each other. The module consists of a set of bounded

[1] https://github.com/randriu/synthesis
[2] https://github.com/moves-rwth/storm
[3] https://github.com/moves-rwth/stormpy

variables spanning the state space of a model. Transitions are handled by the guarded commands in the following form:

$$\texttt{guard} \rightarrow p_1 : \texttt{update}_1 + \cdots + p_n : \texttt{update}_n.$$

The guard represents a Boolean expression over the module variables. Evaluating the guard to a logical value `true` evolves the module into one of its successor states, which is achieved by a simple variable update. The probability distribution over expressions $p_1, \cdots, p_n$ deduces the update of a variable. Guards overlapping produces nondeterminism and hence it is not allowed. Essentially, a program $\mathbf{P}$ is a tuple of variables and commands. For a program $\mathbf{P}$ the underlying MC $[\mathbf{P}]$ is an MC that functions the same way is $\mathbf{P}$ on the state level. The program $\mathbf{P}$ satisfies the specification φ iff $[\mathbf{P}] \models \varphi$.

A sketch is an incomplete program containing holes. Holes are the program unfixed parts that need to be assigned from finite set of options. Holes are declared as:

$$\texttt{hole}\ h\ \texttt{either}\ \{\texttt{expr}_1, \cdots, \texttt{expr}_2\},$$

where h is the hole ID and $\texttt{expr}_i$ stands for an expression over the program variables. Hole can be used anywhere in the command or variable declaration, as well as part of the guard or an update expression. Assigning all the available holes in a sketch yields a specific program. The synthesis methods described in the previous chapter are used exactly to find such a specific assignment to satisfy certain properties in a reasonable time.

4.2 PAYNT for POMDPs and FSC synthesis

As already mentioned, PAYNT was originally designed to perform synthesis of probabilistic programs. We already showed the connection between the synthesis of FSCs and MCs, then how the MCs synthesis is delivered in practice in the form of the synthesis of sketches of probabilistic programs, and now to enclose the circle, we show how these exact same methods are applied in POMDPs and FSCs synthesis field. Based on the article [3] that originally introduced the use of inductive synthesis methods for POMDPs, respectively, for FSC synthesis, the PAYNT tool was extended to enable the synthesis of FSCs for POMDPs.

In this synthesis mode, the input sketch has no holes as it instead represents the actual POMDP for FSC synthesis. Holes are, however, still used during the synthesis process, but they do have a different semantic meaning. Sketch defines all the necessary parts of the POMDP such as states, actions, transition functions (matrices), and observations. The synthesis itself then consists of two parts. In the first part, the design space of a set memory size is created, which includes all the possible FSCs, in the second part this created design space is then explored in order to obtain the best suitable FSC. Memory size specifies the number of memory nodes that the synthesized FSC should contain. The exploration of the design space is carried out using the synthesis methods introduced in Chapter 3.

The program utilizes a list of Python classes which represent individual holes within a design space. These classes have properties including `name`, `options`, and `option_labels`. The name of a hole is a string that encodes the memory values of FSCs and observations received by the agent from POMDP. The name follows a structure of `"T([O], M)"`, where `T` specifies the type of hole (`A` for action, `M` for memory, or `AM` for the combination of both), `O` denotes the observation as a string, and M represents the numerical memory value of the node. The „options“ attribute of the hole class is a list of integers that correspond to possible outcomes that may occur next, given the received observation. The specific

outcome depends on the type of hole, which could be an action to take, a memory update, or a combination of both.

4.2.1 FSC synthesis strategies

Apart from the classic synthesis of FSCs for POMDPs, PAYNT is also able to employ different strategies during synthesis to obtain better results.

Iterative Strategy

First strategy is called a iterative strategy. This strategy is based on a very simple approach, as essentially it just iteratively increases the memory size, and thus allowing one to find FSCs with more memory nodes in a larger design space. This strategy is guaranteed to find the best available FSC for each memory size at the cost of a longer execution time as the design space grows significantly in size.

Memory Injection Strategy

The second strategy is called memory injection strategy and was introduced in article [3]. It leverages the information from previous iterations of the design space exploration and based on these it adds (injects) the memory to inconsistent observations. Then it uses the symmetry removal approach to shrink the design space again by removing the FSCs with the same values.

Chapter 5

Greedy construction of CEs for MDP

In previous chapters, inductive synthesis methods such as CEGIS or AR were described as a possible means to synthesize an FSC for POMDP that exhibits certain behaviors to satisfy given specifications. In this chapter, we will focus on the CEGIS synthesis method and, more precisely, on the types of CEs and the approaches of creating such CEs. Generally speaking, as already mentioned in Section 3.3, CEGIS works with MCs, it constructs the CEs as critical subsystems, a fraction of the original MC, and from the subsystems it infers the conflicts which then prunes the set of not yet analyzed realisations accordingly. From now on we will focus on more general CEs, that is, the MDP based CEs, to explore if there is a potential to obtain smaller number of conflicts and thus prune more realisations from the set of all realisations.

5.1 Counterexamples for MDP

This section is inspired by the article [12]. The idea of CE for MDP is somewhat analogous and yet different from the concept of CE for MCs explained in Subsection 2.1.2. Again, let us have some MDP $M = (S, s_0, Act, \mathcal{P})$ and a specification $\varphi \equiv \mathbb{P}_{\bowtie\lambda}[F\, T]$ that is rejected. Similarly, as with MCs, certain actual execution of a model, which violates the specification φ and serves as an CE, would be beneficial here. We intentionally omit CEs in form of the set of paths and focus on the critical subsystems. Path approach is analogous with the MCs however instead of using the BSCC, its counterpart for MDPs called maximum end component is used. In the following definition we show a subsystem lifted for MDP.

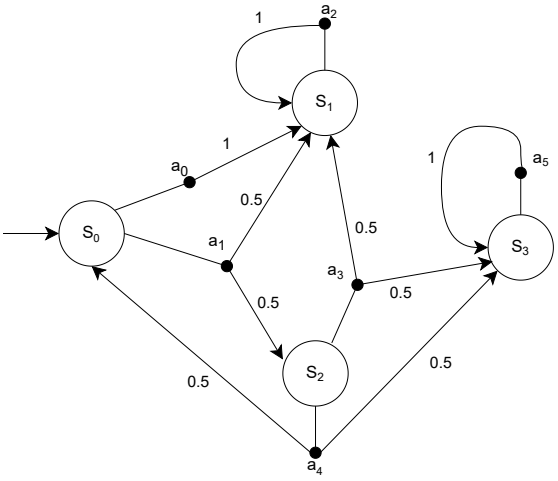
Definition 5.1. *Let $M = (S, s_0, Act, \mathcal{P})$ be an MDP, state $s_\perp$ such that $s_\perp \notin S$ and $C \subseteq S$ such that $s_0 \in C$. The sub-MDP wrt. C is an MDP $M{\downarrow}C = (C \cup \{s_\perp\}, s_0, Act', \mathcal{P}')$ where the set of actions Act' and partial transition probability function $\mathcal{P}'$ are defined as follows:*

$$Act' = \{a \in Act | \exists s \in C \ .a \in Act(s)\} \cup \{a_\perp\},$$

$$\mathcal{P}'(s,a,s') = \begin{cases} \mathcal{P}(s,a,s') & \text{if } s,s' \in C \text{ and } a \in Act', \\ 1 - \sum_{s'' \in S \setminus C} P(s,a,s'') & \text{if } s \in C \text{ and } a \in Act' \text{ and } s' = s_\perp, \\ 1 & \text{if } s = s' = s_\perp \text{ and } a = a_\perp, \\ 0 & else. \end{cases}$$

The problem here is the nondeterminism in the MDP model since there may be multiple outcomes based on the actions that are chosen. Minimizing and maximizing schedulers $\sigma_{min}, \sigma_{max}$ together with their respective $\mathbb{P}_{min}$ and $\mathbb{P}_{max}$ from Subsection 2.2.1 comes in handy.

Definition 5.2. *Let $M = (S, s_0, Act, \mathcal{P})$ be an MDP, $\varphi \equiv \mathbb{P}_{\leq\lambda}[FT]$ and $\varphi' \equiv \mathbb{P}_{\geq\lambda}[FT]$ be a safety and liveness properties, such that both $M \not\models \varphi$ and $M \not\models \varphi'$ hold. Then if for some C holds that MC induced by the minimizing scheduler $M{\downarrow}C^{\sigma_{min}} \not\models \varphi$, then the set C and the corresponding subsystem $M{\downarrow}C$ are called critical. Analogously for liveness property, if for some C holds that MC induced by the maximizing scheduler violates a modified specification φ' such that $M{\downarrow}C^{\sigma_{max}} \not\models \mathbb{P}_{\geq\lambda}[F\, T \cup \{s_\perp\}]$, then we talk about the critical set C and its corresponding subsystem $M{\downarrow}C$. Also, if $|C| \leq |C'|$ for every C' then we call it a minimal critical subsystem. For simplification we can talk about the minimal $\mathbb{P}_{min}$ and maximal $\mathbb{P}_{max}$ probability.*



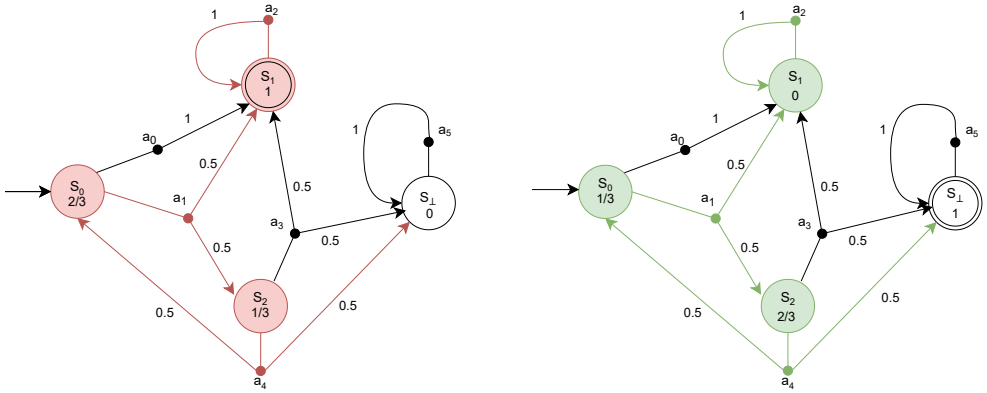
(a) MDP graph.

$$\begin{array}{c} (s_0,a_0) \\ (s_0,a_1) \\ (s_1,a_2) \\ (s_2,a_3) \\ (s_2,a_4) \\ (s_3,a_5) \end{array} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0.5 & 0.5 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0.5 & 0 & 0.5 \\ 0.5 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(b) MDP transition probability matrix for individual actions.

Figure 5.1: Basic MDP with 4 states and 6 actions for the purpose of MDP CEs demonstration.

Example 5.1. *Now, let us briefly showcase the CEs for MDP in the form of critical subsystems. Let $M = S, s_0, Act, \mathcal{P}$ be an MDP from Figure 5.1. In Figure 5.2a the critical states are colored red and the actions taken by the minimizing scheduler σ_{min} are colored red as well. A critical set of states $C = \{s_0, s_1, s_2\}$ for a safety property $\mathbb{P}_{\leq 0.6}[F\,\{s_1\}]$ induces a critical subsystem (sub-MDP) $M{\downarrow}C$. By obtaining the minimizing scheduler σ_{min} and the corresponding lower bound $\mathbb{P}_{min}$, then it is clear that the induced MC $M{\downarrow}C^{\sigma_{min}}$ violates the the safety property $\mathbb{P}_{\leq 0.6}[F\,\{s_1\}]$ and all states from the set $S \setminus C$ which are not present in the sub-MDP are replaced by the sink state $s_\perp$ which acts as a common reroute. Similarly in Figure 5.2b for the liveness property $\mathbb{P}_{\geq 0.75}[F\{s_3\}]$ the CE is depicted as well. This time the critical set of states $C' = s_0, s_1, s_2$ and the actions chosen by the maximizing scheduler σ_{max} are colored green. Induced critical subsystem $M \downarrow C'$ together with the maximizing scheduler σ_{max} induces MC $M{\downarrow}C'^{\sigma_{max}}$ violates the modified property $\mathbb{P}_{\geq 0.75}[F\,\{s_1\} \cup \{s_\perp\}]$. Adding the sink state to the set of target states follows the same logic as in Example 2.3.*



(a) CE for an MDP from Figure 5.1 and safety property $\mathbb{P}_{\leq 0.6}[F\ \{s_1\}]$.

(b) CE for an MDP from Figure 5.1 and liveness property $\mathbb{P}_{\geq 0.75}[F\ \{s_3\}]$.

Figure 5.2: Liveness and safety property CEs for MDP from Figure 5.1.

There are various approaches to construct such critical subsystems. In the next sections, we will take a closer look at some existing approaches and also outline a greedy method for such construction.

5.2 Existing approaches for MDP CEs construction

First we take a closer look at some existing approaches to construct CEs for MDP. A program called Small WITnessing SubSystems[1] (SWITSS) introduced by Jantsch et. al. [11] and based on the theory of Farkas certificates for lower and upper bounds on minimal and maximal reachability probabilities in MDP described by the Funke et. al. [12].

The witnessing subsystem (witness) is basically a counterpart of CEs which serves as a diagnostic information on why a reachability property holds. The SWITSS program is capable of creating such witnessing subsystems based on the translation between the witnessing subsystems and Farkas certificates. Without going into too much detail, based on the reduction of the problem to a mixed integer linear programming (MILP), SWITSS implements the exact and heuristic approaches to yield the witnessing subsystems for some concrete specification. Such witnesses are ideal candidates to be used as CEs in CEGIS technique, and thus the goal here is not to fully understand the SWITSS approach but rather to consider it as a form of black box that is capable of the construction of CEs. Leveraging the common interface provided by the STORM model checker, we are able to integrate SWITSS as a third party CE generator. PAYNT itself is built on top of the STORM and despite having its own representation for probabilistic models such as MC, MDP, POMDP, etc., it is at the same time capable of using the STORM models representation. SWITSS is then able to convert the STORM (more precisely the Stormpy models) into its own internal representation, and thus the interface bridge exists between these programs.

SWITSS tool is written in Python 3 and Cython programming languages and it makes use of the common Python 3 libraries like Numpy[2] or SciPy[3]. SWITSS also uses the

[1] https://github.com/simonjantsch/switss
[2] https://numpy.org/
[3] https://scipy.org/

PRISM and PuLP solver[4]. Since both SWITSS and PAYNT are mostly written in Python 3 the integration was quite straightforward as it is possible to make the SWITSS calls directly in the PAYNT code, and thus it essentially involved just converting the models from STORM to SWITSS representation, creating and calling the particular witness solvers, which then would yield the small witnessing subsystem, which would be considered CE for some specification φ. Also, two obstacles were encountered during the integration, which will be described later on.

From the design perspective, PAYNT program, more precisely the CEGIS loop, was changed that way that it is able to plug in various CEs generators if they follow the specified interface instead of having just one possible instance. It is now possible to choose the desired CE generator from the command line by just specifing the name, we currently have `storm`, which will be described in more detail in the next Section 5.3, and `switss` CE generators. Schema of the integration is depicted in Figure 5.3.

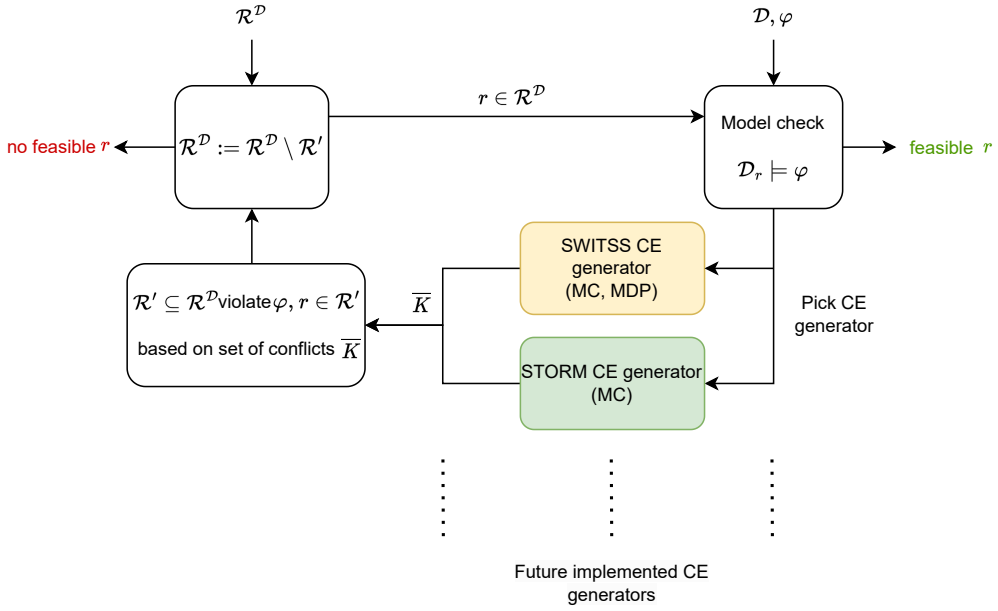


Figure 5.3: Schema of the SWITSS CE generator module integrated into CEGIS loop.

Integration allowed us to use SWITSS to construct both MC and MDP based CEs. As already outlined, SWITSS has its limitations. The first limitation is that it is not able to create CEs for reward-based properties $\varphi \equiv R_{\bowtie\lambda}[F\ T]$. This problem was not addressed because CE construction was still possible on reachability properties and mainly because the results from Section 5.4 rendered SWITSS unusable and it would require one to venture deeper into a SWITSS codebase. The second limitation showed that SWITSS is not able to construct CEs for liveness properties, such as $\varphi \equiv \mathbb{P}_{\geq\lambda}[F\ T]$. This problem was possible to address on the PAYNT side by transforming the liveness property into a safety property with flipped threshold λ, such as $\varphi' \equiv \mathbb{P}_{<1-\lambda}[F\ T']$, compute all BSCCs or maximum end components in the case of MDP, which then would be collapsed each into one new state and such states would be proclaimed the new target set of states T'. After such transformation, the model could be passed to SWITSS solver and the resulting CE would be created. It is crucial to label the states that originated from collapsed BSCCs/maximum end components

[4] https://pypi.org/project/PuLP/

by the number of the original states that these new states replaced, and also it is crucial to keep all the information about the original model such as labels and sketch hole indices. In Section 5.4 we present the experimental evaluation of the SWITSS CEs for both MCs and MDPs and we put them into contrast with the current greedy method implemented in PAYNT on top of the STORM for construction of MC CEs.

5.3 Greedy method

As already outlined in the previous Section 5.2 and confirmed in the following Section 5.4, SWITSS is not a suitable candidate for constructing CEs based on MC or MDP as it suffers from a massive time overhead and incompleteness of the reward property. Portion of the time overhead might be probably mitigated by very nontrivial changes in the SWITSS codebase, however, for the time being the ideal solution would be to adopt some greedy heuristic method for MDP CEs instead of exact solutions obtained by the SWITSS MILP solvers.

The current implementation of PAYNT uses the greedy approach described by Adriuschenko et. al. [2] designed to compute CEs providing small conflicts. The key idea of this procedure, described very briefly, is based on a gradually expanding states of an MC, which are associated with relevant parameters. We define a new greedy method in this section, which is based on the very same approach just outlined, by modifying it to operate on MDPs.

Proposition 5.1. *Recall a realisation of a family of MCs from Definition 3.3. By fixing each parameter based on a realisations we induce an MC $\mathcal{D}_r$. Generalizing this idea, we may only partially fix the realisations. Let $\mathcal{D} = (S, s_0, K, \mathcal{B})$ be a family of MCs and $L \subseteq K$ be a set of parameters. A partial realisation of a family D of MCs with reference to some realisation r is a function $r^L : K \setminus L \to S$ such that $\forall k \in K \setminus L : r^L(k) \in T_k$. Partial realisation r^L induces MDP $\mathcal{D}_{r^L} = (S, s_0, \mathcal{R}^{\mathcal{D}}, \mathcal{P})$, where:*

$$\mathcal{P}(s)(r') = \begin{cases} \mathcal{B}_{r'} & \text{if } L \cap supp(\mathcal{B}(s)) \neq \emptyset, \\ \mathcal{B}_{r'} & \text{if } L \cap supp(\mathcal{B}(s)) = \emptyset \text{ and } r' = r, \\ undefined & \text{else.} \end{cases}$$

In other words we fix each parameter from set $K \setminus L$ and leave each parameter from the set L unfixed, which leaves us with a structure similar to the quotient MDP defined in Definition 3.7. In the Figure 5.4 we may observe such induced MDP from partial realisation.

Let us assume that $\mathcal{D}_{r^L}$ violates a reachability property $\varphi \equiv \mathbb{P}_{\bowtie}[FT]$. We first fully define the concept of rerouting which was already outlined in Definition 2.4 and Definition 5.1.

Definition 5.3. *Let MDP $M = (S, s_0, Act, \mathcal{P})$ with states $s_\top, s_\bot \notin S$, set of expanded states $C \subseteq S$ and a rerouting vector $\gamma : S \setminus C \times Act \to [0, 1]$. The rerouting of MDP M with reference to the set C and the vector γ is MDP $M \downarrow C[\gamma] = (S \cup \{s_\top, s_\bot\}, s_0, Act \cup \{a_{\top\bot}\}, \mathcal{P}^C_\gamma)$ where $\mathcal{P}^C_\gamma$ is defined as follows:*

$$\mathcal{P}^C_\gamma(s) = \begin{cases} \mathcal{P}(s) & \text{if } s \in C, \\ [(a_{\top\bot}, s_\top) \to \gamma(s), (a_{\top\bot}, s_\bot) \to (1 - \gamma(s))] & \text{if } s \in S \setminus C, \\ [(a_{\top\bot}, s) \to 1] & \text{if } s \in \{s_\top, s_\bot\}. \end{cases}$$

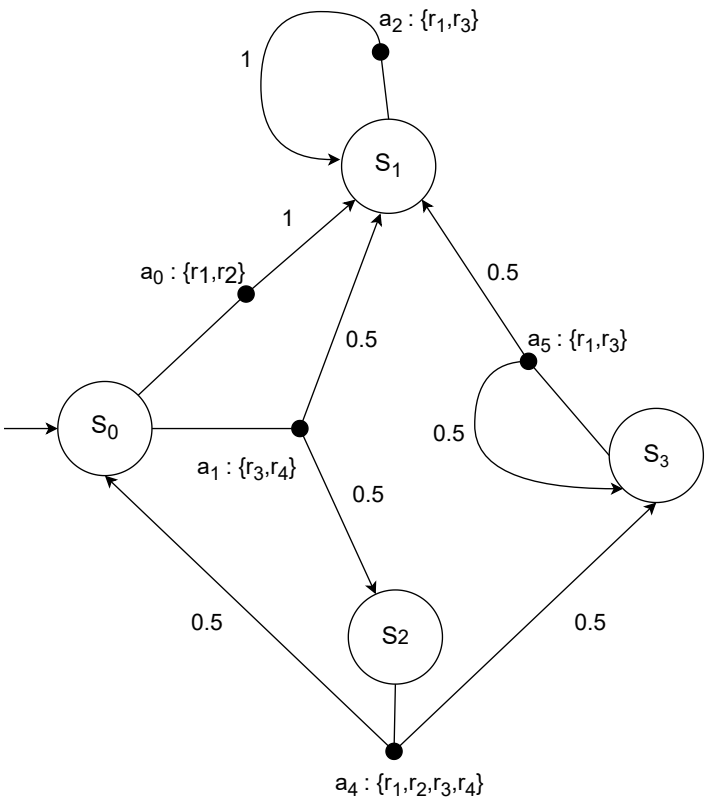
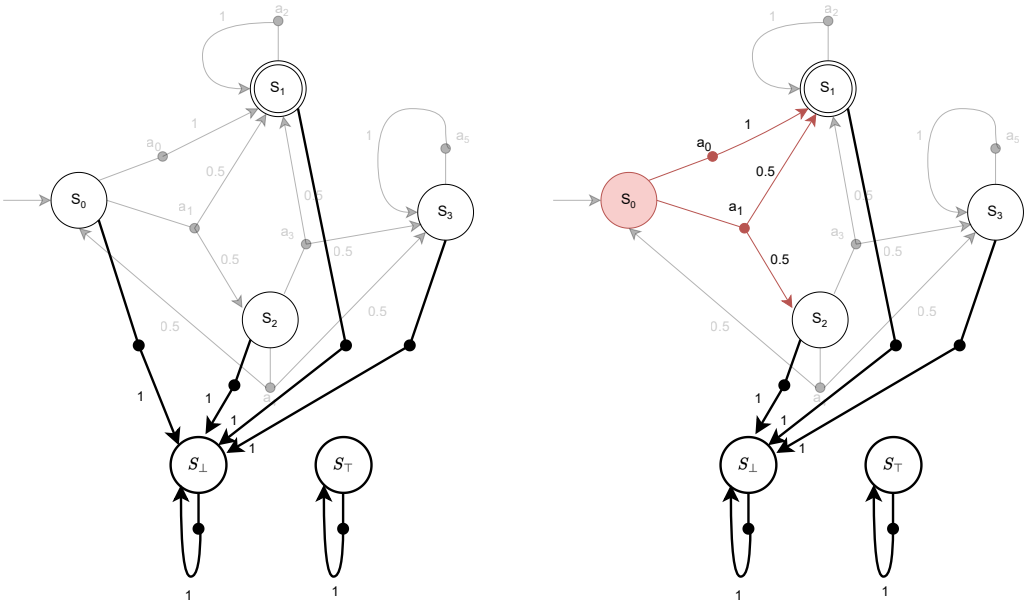


Figure 5.4: Induced MDP from partial realisation r_1^L with the set of unfixed parameters $L = \{X\}$ from the family of MCs depicted in the Figure 3.1.

In other words, $M \downarrow C[\gamma]$ adds two additional sink states $s_\top$ (so-called true sink state) and $s_\bot$ (so-called false sink state) to MDP M, replacing the all actions of any non-expanded state $s \in S \setminus C$ with a single action $a_{\top\bot}$ which leads to sink state $s_\top$ with the probability $\gamma(s)$ and to the sink state $s_\bot$ with the probability of $1 - \gamma$. True sink state $s_\top$ is also needed to be added among the target states for the sake of handling the liveness properties as already shown in the Definition 2.5, although in the mentioned definition we kept the idea simplified to have only one sink state at a time. Rerouted transitions via actions may be viewed as shortcuts, and thus, the transition is possible to be carried out by just skipping the successor state completely and moving to one of the sink states directly. In classic CEGIS the rerouting vector is either $\gamma = 0$ for safety properties or $\gamma = 1$ for liveness properties, however when using CEGIS in hybrid synthesis, rerouting vectors is based on lower and upper bounds obtained from the AR.

For obtaining the CE the state expansion (or exploration) is used. First, we start with the initial set $C_0 = \emptyset$ where all states are initially rerouted. We check the if the rerouted MDP $M \downarrow C_0[\gamma]$ is the CE and if so, the construction process terminates. Otherwise if $M \downarrow C_0[\gamma]$ satisfies the property, the state space needs to be expand with some new state from $s \in S$ that is particulary reachable from the already explored states. Expanding the states essentially means that the rerouting is removed and the states retains its original actions, and thus outgoing transitions. Such rerouting for the safety property φ is depicted in the Figure 5.5.



(a) Set $C = \emptyset$, all states are rerouted. (b) Set $C = \{s_0\}$, rest of the states is rerouted.

Figure 5.5: Construction of CE to MDP from Figure 5.1 for safety property $\mathbb{P}_{\leq 0.6}[F\ \{s_1\}]$ using the rerouting vector $\gamma = 0$.

We now explain the Algorithm 4 which is the modified version of finding the CE lifted for MDPs. To choose a best suitable state for the expansion, we should aim to obtain such conflicts which contains low number of the relevant parameters. We keep the same approach as Andriushchenko et. al. [2] suggests. That is expanding multiple states at once based on the set of relevant parameters $\overline{K}$. Starting with the empty set of such relevant parameters $\overline{K}_0 = \emptyset$, only those states that are currently reachable from the initial state s_0 through the successors states which are not associated with any irrelevant parameter (or hole from the program/sketch point of view). State horizon H_i contains states that are possible to be considered for the expansion as they are reachable from the set of states C_i but they contain at least one irrelevant parameter. Constructing the rerouting $M \downarrow C_i[\gamma]$ and checking it against the property φ with the possibility of yielding the MDP CE follows. If the rerouted MDP is still not a CE, we pick some state s_{expand} from the state horizon with the lowest number of irrelevant parameters and we add those parameters to the conflict $\overline{K}$.

Algorithm 4: Greedy MDP counterexample construction based on rerouting.

Input: An MDP $\mathcal{D}_{r^L}$, a property $\varphi \equiv \mathbb{P}_{\bowtie\lambda}[F\,T]$ such that $\mathcal{D}_{r^L} \not\models \varphi$ and a rerouting vector γ.

Output: A conflict $\overline{K}$ for MDP $\mathcal{D}_{r^L}$ and property φ

ConstructConflict $(\mathcal{D}_{r^L}, \varphi, \gamma)$:

- $i := 0$
- $\overline{K}_i := \emptyset$
- **while** *true* **do**
  - $C_i, H_i := reachableViaRelevantParameters(\mathcal{D}_{r^L}, \overline{K}_i)$
  - $D_i := \mathcal{D}_{r^L} \downarrow C_i[\gamma]$
  - **if** $\mathbb{P}_{min}[D_i \models \varphi] > \lambda$ ***and*** $\bowtie \in \{<, \leq\}$ **then**
    - **return** K_i
  - **if** $\mathbb{P}_{max}[D_i \models \varphi] < \lambda$ ***and*** $\bowtie \in \{>, \geq\}$ **then**
    - **return** K_i
  - $s_{expand} = pickToExpand(H_i, \overline{K}_i)$
  - $\overline{K}_{i+1} := \overline{K}_i \cup supp(\mathcal{B}(s_{expand}))$ $i := i + 1$
- **return** $\overline{K}_i$

5.4 Experimental evaluation of SWITSS CEs

In this section we take a closer look at the experimental evaluation of the CEs constructed by SWITSS, both MC and MDP variants, and we compare them with the current greedy method implemented in PAYNT on top of the STORM, which produces the MC CEs. Before the experiments, we need to ask the main questions to which we would like to find answers by conducting these experiments, those are:

1) What is the quality of the size and construction speed of CEs constructed by SWITSS compared to the current approach?

2) Is the SWITSS program a viable tool to march onward with for the MDP CEs constructing?

5.4.1 Experiments settings

Since we needed to obtain the direct side-by-side comparison of the STORM and SWITSS CEs, we would need to run both of them inside a single CEGIS loop during the single execution of the PAYNT. This is because the way CEGIS picks a realisation of a family for model checking is nondeterministic, and each time the PAYNT program is executed, there may be a different realisation chosen. Thusforth, the experiment is conducted in a way that each iteration the CEGIS loop calls the CE construction twice, both STORM and the SWITSS. In order for CEGIS to progress forward, we always just save the statistics from both CE generators and continue with the set of conflicts provided by the STORM.

In these experiments, we are interested in the average conflict size, which is the number of relevant holes (parameters) that are contained by the critical subsystem, the average time per conflict construction in seconds, respectively, construction of CE, as the conflicts are inferred from the CE and the total time spent constructing the CEs. We apply a roughly 600 second timeout for the synthesis, and thus if the PAYNT is not able to find a

feasible assignment or does not proclaim the property unjustifiable in the concrete model, the synthesis is terminated. That being said, some of the data times from the data tables in the following sections might imply that the timeout threshold was much higher and that is due to the fact that the termination was performed at the PAYNT program level, once the STORM or SWITSS returned the program control.

Lastly, the experiments were carried out on a set of the latest POMDP benchmark models for the integration of STORM from the PAYNT[5] repository and the specification of the experiments was the optimality property $\mathbb{P}_{max}$. FSC synthesis was performed in the mode for searching 1-FSCs, none of the iterative or memory injection strategies from Subsection 4.2.1 were used.

5.4.2 Results of SWITSS MC CEs

In Table 5.1 we may observe the initial experiments of the STORM MC CEs against the SWITSS MC CEs. It is clear from the table that, from the conflict size perspective, both generators perform comparably. For some models like `Drone-8-2` SWITSS delivers the smaller conflicts, on the other hand for some models STORM gives us the better CEs. Generally, this comparison may be neglected. The key metric here is the average time per conflict construction and the total time of the CEs constructions as in this field the SWITSS suffers from massive time overhead. In some cases, in the Table 5.1 highlighted by the red color, the SWITSS performs almost 200 worse than the STORM.

Model	Avg conflict size		Avg time per conflict [s]		Total CE construction time [s]	
	STORM MC	**SWITSS MC**	**STORM MC**	**SWITSS MC**	**STORM MC**	**SWITSS MC**
Drone-4-1	15.639	21.153	0.026	3.342	3.766	481.292
Drone-4-2	12.358	14.705	0.009	1.727	2.315	438.678
Drone-8-2	41.438	23.938	0.191	10.770	3.049	172.317
Grid-avoid-4-0	1.000	1.000	0.001	0.039	0.003	0.156
Grid-avoid-4-0.1	1.000	1.000	0.001	0.041	0.003	0.165
Grid-large-30-5	19.126	19.126	0.004	0.209	11.496	538.629
Refuel-06	11.305	9.802	0.004	0.115	18.649	518.050
Refuel-08	13.151	12.369	0.005	0.213	12.377	540.843

Table 5.1: Comparison of the STORM MC CEs and SWITSS MC CEs by the average conflict size, average conflict construction time and total time spent with CEs construction. Row highlighted with the red color shows the case where SWITSS performs the worst from the time perspective.

5.4.3 Results of SWITSS MDP CEs

As our main goal was to enable MDP CEs for CEGIS in PAYNT and comparing just the ability of SWITSS to construct MC CEs would not be objective, we performed the experiments for the MDP CEs as well, although results from Subsection 5.4.2 already predict similar behavior from the time and size perspective. In the Table 5.2 we can see that average

[5] https://github.com/randriu/synthesis

conflict size fluctuates more or less in the same manner as already observed in the SWITSS MC CEs experiments; this time we highlight for example models such as `Refuel-06` and `Refuel-08` where SWITSS delivers the smaller conflicts. Unfortunately, time duration performance is even worse in this case. For one particular model `Drone-8-2` highlighted again with the red color in Table 5.2 we observe insanely large difference between the STORM and SWITSS, which is roughly 565 000× more spent with only one CE construction. This is also the reason why the timeout limit was exceeded so drastically from 600 seconds to almost 2 hours, because the first CE was constructed inside the SWITSS for such a long time and only after that the program control was returned to PAYNT, which immediately terminated.

Model	Avg conflict size		Avg time per conflict [s]		Total CE construction time [s]	
	STORM MC	SWITSS MDP	STORM MC	SWITSS MDP	STORM MC	SWITSS MDP
Drone-4-1	12.214	64.571	0.017	44.082	0.235	617.150
Drone-4-2	9.900	64.200	0.005	60.333	0.049	603.332
Drone-8-2	9.000	49.000	0.010	5653.110	0.010	5653.110
Grid-avoid-4-0	1.000	1.000	0.001	0.037	0.002	0.147
Grid-avoid-4-0.1	1.000	1.000	0.001	0.039	0.002	0.154
Refuel-06	11.672	5.912	0.004	0.146	16.014	532.225
Refuel-08	13.602	9.971	0.007	0.634	6.366	570.519
Refuel-20	29.479	29.688	0.043	12.293	2.040	590.051

Table 5.2: Comparison of the STORM MC CEs and SWITSS MDP CEs by the average conflict size, average conflict construction time and total time spent with CEs construction. Row highlighted with the red color shows the case where SWITSS performs the worst from the time perspective.

Additionally experiments were also conducted with the MDP CEs from the greedy generator which principle was outlined in the Section 5.3 and will be described with more context in Chapter 6. Results are depicted in Table 5.3. Average sizes of the conflicts vary depending on the models and both sides have their ups and downs, however the significant time overhead of the SWITSS CE generator is still massively visible. For model `Drone-8-2`, which is highlighted red in Table 5.3 STORM construct a larger conflict but over 200 times faster than SWITSS. POMDPs `Drone-4-1` and `Drone-4-2` hightlighted green are interesting cases where STORM greedy MDP CE generator delivers much smaller conflicts in the fraction of SWITSS time.

Model	Avg conflict size		Avg time per conflict [s]		Total CE construction time [s]	
	STORM MDP	**SWITSS MDP**	**STORM MDP**	**SWITSS MDP**	**STORM MDP**	**SWITSS MDP**
Drone-4-1	24.000	64.571	0.705	44.082	9.872	617.150
Drone-4-2	23.000	64.200	0.525	60.333	5.247	603.332
Drone-8-2	80.000	49.000	23.577	5653.110	23.577	5653.110
Grid-avoid-4-0	1.000	1.000	0.001	0.037	0.002	0.147
Grid-avoid-4-0.1	1.000	1.000	0.001	0.039	0.003	0.154
Refuel-06	12.361	5.912	0.009	0.146	32.416	532.225
Refuel-08	16.609	9.971	0.019	0.634	17.363	570.519
Refuel-20	35.854	29.688	0.179	12.293	8.572	590.051

Table 5.3: Comparison of the STORM MDP CEs and SWITSS MDP CEs by the average conflict size, average conflict construction time and total time spent with CEs construction.

From both experiments it is clear that the SWITSS is not a suitable candidate for MDP CEs generator because it suffers from massive time performance overhead, it delivers comparable results, and it supports less features for the synthesis purpose.

Chapter 6

Using CEs for CEGIS

In the previous chapter, the greedy method for constructing CEs for MDP based on the greedy algorithm introduced by Andriushchenko et. al. [2] is outlined. In this chapter, we focus on more practical usage of the introduced algorithm. As already shown, we are now able to construct the CEs for MDPs, but where do we obtain such MDP which we would be able to construct CE against? In the following sections we explore the so-called simple hole (parameter) generalization which yields us the needed base MDP for the CE construction. Next we describe the key ideas behind the integration of such MDP CEs into PAYNT program as a new CE generator module side by side with the current implementation of the greedy method for MC CEs and SWITSS.

6.1 Simple holes generalization

Recall a partial realisation of a family r^L introduced in Proposition 5.1 which induces MDP D_{r^L}. PAYNT CEGIS loop is already capable of picking a realisation and inducing MC from such realisation by fixing all the holes (parameters). The task here is to identify the set of holes L that may remain unfixed and thus would form a partial realisation.

Definition 6.1. *Let $\mathcal{D} = (S, s_0, K, \mathcal{B})$ be a family of MCs. Let $K' \subseteq K$ be a set of simple (trivial) parameters such that $\forall k \in K' : \exists s \in S, k \in supp(\mathcal{B}(s)) \wedge \forall s' \in S \setminus \{s\} : k \notin supp(\mathcal{B}(s'))$. In other words set K' is a set of all simple parameters that are associated with one single state at maximum.*

Example 6.1. *The family of MCs of Figure 3.1 is a clear example of a family with two parameters X, Y where the parameter X is associated only with the state s_0 and thus the set of simple parameters is defined as $K' = \{s_0\}$.*

Now when the concept of simple parameters is defined, we show how this connects to the partial realisation and induced MDP.

Proposition 6.1. *Let $\mathcal{D} = (S, s_0, K, \mathcal{B})$ be a family of MCs, and let K' be a set of simple parameters. Such a set of simple parameters is a good candidate for a set of parameters L for partial realization. By putting a $L = K'$ we basically generalize those parameters. Let us quickly return to Algorithm 4. We are particularly interested in the set of relevant parameters K_i and the state expansion technique. By generalizing all the simple parameters and fixing all the non-simple parameters (parameters associated with at least 2 states) we are able to rule out all the simple parameters from the set of relevant parameters K_i and*

thus potentially reducing the size of a conflict. The key idea of why generalizing the simple parameter is possible is based on a sole fact that it is associated only with a single state and thus making any choice in that particular state will not change the choice in any other state throughout the whole model.

During the synthesis of FSCs for POMDPs a lot of recent models show the common behavior of having a large number of simple parameters, and hence we explore the ways of generalizing such parameters. Having introduced the concept of single parameters and their important role in inducing an MDP from the family of MCs we now show in the next section how this idea was programatically incorporated into the PAYNT program.

6.2 MDP CEs integration to PAYNT/STORM

In this section we will take a closer look at how Algorithm 4 together with the Proposition 6.1 of generalization of simple holes may be implemented in the PAYNT.

PAYNT in general is written in Python 3 but it frequently uses STORM written in C++. Existence of the Stormpy bindings allows one to call the STORM C++ code directly from the PAYNT Python 3 code. On the PAYNT side all the necessary parsing, model building, and synthesis control is happening, STORM on the other hand provides model checking features and the greedy algorithm introduced by Adriuschenko et. al. [2], which is integrated into STORM as an additional module. To be more precise, it is not implemented in the original STORM GitHub repository[1] but rather in the modified fork[2] of the mentioned repository, which additionally provides convenient features for synthesis of probabilistic programs in PAYNT. One of the many reasons why the greedy method state exploration is located on the STORM side and not on the PAYNT side is definitely the overall speed advantage of the C++ programs, and thus we choose to implement the greedy algorithm for MDP CEs outlined in Algorithm 4 mainly in STORM as well. That being said, it is important to mention that code base for the MDP CEs is not written from scratch but it adapts the already existing code for the MC CEs. In the adapted Figure 6.1 we may observe how already 3 CE generators (STORM MC, STORM MDP and SWITSS) exist in the PAYNT. One may choose from the CE generators available for the CEGIS method when executing the PAYNT.

Algorithm 5 depicts the basic idea of how analysis of a single realisation in CEGIS loop is implemented in PAYNT with the use of MDP CEs. First, we identify the set of simple holes (parameters) based on Definition 6.1. Then we construct a partial realisation r^L using the obtained set of simple holes and by fixing the partial realisation we induce the MDP $\mathcal{D}_{r^L}$. By model checking $\mathcal{D}_{r^L}$ we either obtain the satisfying realisation or if the property is unsatisfied we pick the rerouting vector based on a type of property (safety or liveness) and initiate the conflict construction procedure (line 10). Everything until this point happens on the PAYNT side, once the conflict construction is invoked, STORM takes over the program control. Conflict construction is carried away by the slightly modified version of Algorithm 4. First change is that the algorithm now accepts one additional input in the form of a set of simple holes L, these holes are added to the list of relevant holes right from the start and thus the state exploration may expand states associated with the simple holes immediately. State exploration then continues in the same way with the one final change on line 24, where in the end from the set of relevant holes (conflict) the set of simple holes

[1] https://github.com/moves-rwth/storm
[2] https://github.com/randriu/storm/tree/synthesis

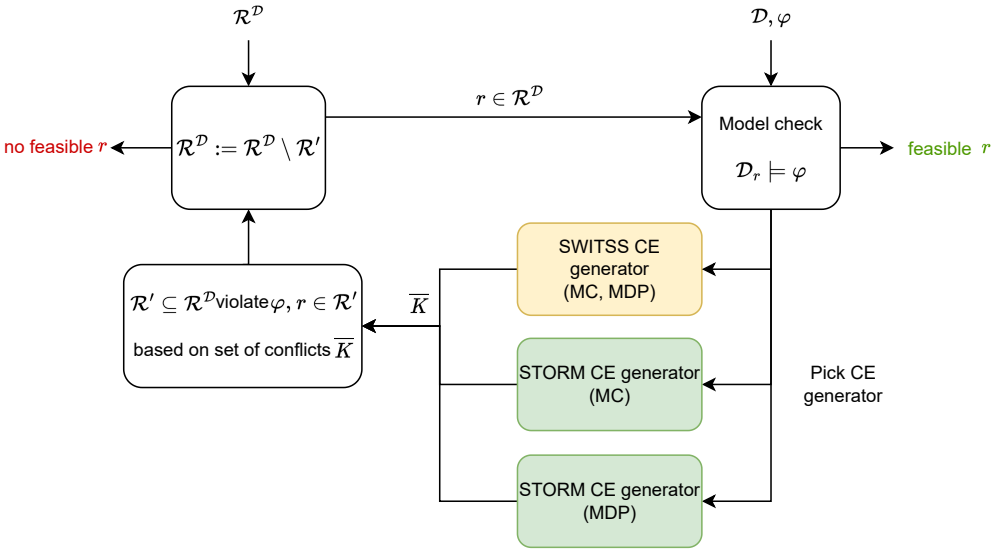


Figure 6.1: Schema of the STORM MDP CE generator module integrated into CEGIS loop.

is subtracted since none of the simple holes, which underwent the generalization process are relevant at this point.

In addition, the process of state exploration during the conflict construction was changed a bit. Currently in the greedy method for MC CEs, the state exploration iterations (waves) are precomputed in advance, and only then the actual model checking of each iteration rerouted MC is carried out. This is, however, ineffective. Since we are always interested in the smallest possible conflict, we do not need to precompute the iterations in advance. Instead, we may just apply the Algorithm 4 literally by alternating between the state exploration phase and the model-checking phase. The newly introduced greedy method for MDP CE construction already refrained from the wave precomputation and in the future the same is possible to apply for the STORM MC CE generator. Moreover, both STORM MC and MDP CE generators are possible in the future with some thorough design to be merged into a single CE generator.

Remark. *It is important to mention that the greedy algorithm for MDP CEs is currently capable of synthesizing specifications containing only single property. Reason behind that is that for multiple properties that the MDP $\mathcal{D}_{r^L}$ satisfies, there may exist different schedulers for the MDP and thus there would not be a unambiguous accepting realisation.*

Algorithm 5: CEGIS loop family assignment analysis using the greedy method for MDP CEs construction based on the generalization of simple holes

Input: A family of MCs $\mathcal{D} = (S, s_0, K, \mathcal{B})$, realisation r of a family $\mathcal{D}$, a property $\varphi \equiv \mathbb{P}_{\bowtie\lambda}[F\ T]$.
Output: A conflict $\overline{K}$ or accepting realisation r

AnalyzeFamilyAssignment *($\mathcal{D}, r, \varphi$):*
- $L := identifySimpleHoles(\mathcal{D})$
- $\mathcal{D}_{r^L}, r^L := fixRealisationPartially(\mathcal{D}, r, L)$
- **if** $\mathcal{D}_{r^L} \models \varphi$ **then**
  - **return** r^L
- **if** $\bowtie \in \{<, \leq\}$ **then**
  - $\gamma := 0$
- **else**
  - $\gamma := 1$
- $\overline{K} := constructConflicts(\mathcal{D}_{r^L}, \varphi, \gamma, L)$
- **return** $\overline{K}$

Input: An MDP D_{r^L}, a property $\varphi \equiv \mathbb{P}_{\bowtie\lambda}[F\ T]$ such that $\mathcal{D}_{r^L} \not\models \varphi$, a rerouting vector γ, set of simple holes L
Output: A conflict $\overline{K}$ for MDP $\mathcal{D}_{r^L}$ and property φ

ConstructConflicts *($\mathcal{D}_{r^L}, \varphi, \gamma, L$):*
- $i := 0$
- $\overline{K}_i := L$
- **while** *true* **do**
  - $C_i, H_i := reachableViaRelevantHoles(\mathcal{D}_{r^L}, \overline{K}_i)$
  - $D_i := \mathcal{D}_{r^L} \downarrow C_i[\gamma]$
  - **if** $\mathbb{P}_{min}[D_i \models \varphi] > \lambda$ ***and*** $\bowtie \in \{<, \leq\}$ **then**
    - **return** K_i
  - **if** $\mathbb{P}_{max}[D_i \models \varphi] < \lambda$ ***and*** $\bowtie \in \{>, \geq\}$ **then**
    - **return** K_i
  - $s_{expand} = pickToExpand(H_i, \overline{K}_i)$
  - $\overline{K}_{i+1} := \overline{K}_i \cup supp(\mathcal{B}(s_{expand}))$ $i := i + 1$
- **return** $\overline{K}_i \setminus L$

6.3 Initial experimental evaluation

Now when the greedy method for generating MDP CEs in PAYNT via STORM is outlined, defined and implemented, we now carry out the experimental evaluation of this initial implementation, where we compare it to the greedy MC CE method. Before the evaluation we again lay down a couple of questions which we would like to find the answer to:

1) What is the quality of the size and construction speed of the CEs constructed by greedy method for MDP CEs compared to the current approach with MCs?

2) How efficiently are MDP CEs able to prune the design space and converge to the optimal solution?

6.3.1 Experiments settings

First experiment settings is very similar to one mentioned in Subsection 5.4.1. STORM MDP and MC CE generators are run side by side, average conflict size, average construction time per conflict, and total time metrics are collected and synthesis timeouts after 600 seconds. The same set of POMDP benchmarks and only the optimality property P_{max} are considered.

The second experiment is then conducted the way that we run 2 separate runs of the synthesis. First run executes the STORM MC CE generator, and second run uses the STORM MDP generator. Timeout is again set to the 600 seconds, and this time a set of POMDP is extended with models where the reward properties are specified. This time we are interested in the synthesis result (optimal value or non-feasibility) and the time spent to converge to such result.

6.3.2 Greedy MC and MDP CEs conflict quality

In the Table 6.1 which shows the initial results of the first experiment, it is clear that the newly created greedy method produces comparable or larger conflicts in the majority of the experiments. The most likely cause of such growth of conflict size is the introduction of new non-simple holes into the conflict by generalizing too many simple holes, we take a closer look at this problem in Section 6.4. On the first glance this might be seen in a not so optimistic way, however, the second metric which we should likely pay attention at this point is the overall state of the synthesis, does it converge to the optimal solutions and how long does it take?

Model	Avg conflict size		Avg time per conflict [s]		Total CE construction time [s]	
	STORM MDP	STORM MC	STORM MDP	STORM MC	STORM MDP	STORM MC
Drone-4-1	24.000	12.214	0.705	0.017	9.872	0.235
Drone-4-2	23.000	9.900	0.525	0.005	5.247	0.049
Drone-8-2	80.000	9.000	23.577	0.010	23.577	0.010
Grid-avoid-4-0	1.000	1.000	0.001	0.001	0.002	0.002
Grid-avoid-4-0.1	1.000	1.000	0.001	0.001	0.003	0.002
Refuel-06	12.361	11.672	0.009	0.004	32.416	16.014
Refuel-08	16.609	13.602	0.019	0.007	17.363	6.366
Refuel-20	35.854	29.479	0.179	0.043	8.572	2.040

Table 6.1: Comparison of STORM MDP CEs and STORM MC CEs by the average conflict size, average conflict construction time and total time spent with CEs construction.

6.3.3 Greedy MC and MDP CEs synthesis results

In this subsection we take a closer look at how MDP CEs actually affect the synthesis result. Table 6.2 shows that the MDP CEs indeed, despite the larger size of the conflicts, positively affect the synthesis. For all three drone models, the MDP CEs were able to find a feasible assignment of the holes compared to the MC CEs with optimality equal to 0.0 after 600 seconds synthesis run. Synthesis of POMDP model `Rocks-12` is able to find some

feasible assignment within the 600 seconds while MC CEs were not able to find any feasible solution at all. Most interesting is then `Maze-alex` as in this case the synthesis time was reduced almost 13 times by employing the MDP CEs.

Model	Property	MC		MDP	
		Result	Time [s]	Result	Time [s]
Drone-4-1	$\mathbb{P}_{max}$	optimal: 0.0	604.57	optimal: 0.401214	600.03
Drone-4-2	$\mathbb{P}_{max}$	optimal: 0.0	607.16	optimal: 0.881402	600.37
Drone-8-2	$\mathbb{P}_{max}$	optimal: 0.0	642.06	optimal: 0.757757	623.63
Grid-avoid-4-0	$\mathbb{P}_{max}$	optimal: 0.214286	0.07	optimal: 0.214286	0.07
Grid-avoid-4-0.1	$\mathbb{P}_{max}$	optimal: 0.214286	0.07	optimal: 0.214286	0.06
Grid-large-20-5	$\mathbb{R}_{min}$	feasible: no	156.35	feasible: no	105.87
Grid-large-30-5	$\mathbb{R}_{min}$	feasible: no	600.08	feasible: no	600.08
Lanes-100-combined-new	$\mathbb{R}_{min}$	optimal: 10241.939783	80.15	optimal: 10241.939783	84.85
Maze-alex	$\mathbb{R}_{min}$	optimal: 71.882276	92.76	optimal: 71.692948	7.36
Network-3-8-20	$\mathbb{R}_{min}$	optimal: 57.5925	602.21	optimal: 57.5925	601.00
Refuel-06	$\mathbb{P}_{max}$	optimal: 0.301668	600.02	optimal: 0.329448	600.01
Refuel-08	$\mathbb{P}_{max}$	optimal: 0.22273	600.00	optimal: 0.199427	600.02
Refuel-20	$\mathbb{P}_{max}$	optimal: 0.06316	600.01	optimal: 0.0	600.08
Rocks-12	$\mathbb{R}_{min}$	feasible: no	600.06	optimal: 374.34908	600.28
Rocks-16	$\mathbb{R}_{min}$	optimal: 223.44875	600.07	optimal: 195.926024	600.62

Table 6.2: Comparison of STORM MC CEs and STORM MDP CEs by the result of the synthesis and elapsed time. Rows with the time above 600 seconds means that the synthesis timeout before the best result was obtained.

6.4 MDP CEs greedy construction variations

As results from the initial implementation of the greedy MDP CEs construction show, the obtained conflicts are larger on average, which might seem a little bit contradictory to the fact that we are generalizing a lot of simple holes. However the generalizations seemingly come with a one drawback. Generalized holes, even though they themselves are not relevant at any possible time, may introduce to the conflict additional revelavant holes by simply reaching more states. This brings the idea of a smarter generalization of simple holes so we do not necessarily generalize all of them but only the chosen portion.

We define the following strategies as attempt to mitigate such behavior:

Apriori simple holes stats

First strategy is based on the collection of statistics prior to the actual run of the synthesis. Before the execution of the CEGIS with the MDP CE generator, a timeboxed run of the CEGIS with the MC CE generator is executed. During this run we collect for each simple hole a number of conflicts that the particular simple hole was part of. In other words, how many times the MC CE generator was unable to generalize the simple hole. Then during the actual execution of CEGIS with MDP CE generator, we compute the average occurrence from all simple hole occurrences and we generalize only those simple holes whose occurrence number is above the average as we are trying to focus on generalizing those holes that the MC CE generator was unable to.

Randomisation

In the second strategy, we suggest a simple concept of an oracle that for each of the simple holes decides whether it should be generalized or not based on randomness. For each hole there is a probability of $R = 0.5$ that the hole will be generalized.

Randomisation based on the hole position

Last strategy is also based on the randomisation, however this time the chance R of a simple hole being generalized starts with the very low value and it grows with the number of iterations that the state space exploration has already made. Such a chance is given by the following equation:

$$R = \frac{i}{i + 1 + |K'|},$$

where i stands for number of current iteration and $|K'|$ is the total number of simple holes.

6.5 Variants experimental evaluation results

In this section we take a closer look at the experimental evaluation of the strategies attempting the smarter simple holes generalization suggested in the previous section. The experimental settings remained the same as for the second experiment in Subsection 6.3.1. In addition the variant of the combined MDP an MC CEs generators was evaluated, which basically just runs both CE generators in series and combined the resulting conflicts together. In Table 6.3 the results of randomised generalization strategies are visualized. POMDP models highlighted with the green colors are the same models from the Table 6.2 which exhibited interesting behavior in the initial evaluation. Overall the randomised approach did not bring any improvement at all. Classic randomisation performs a lot better than the randomisation which takes the hole positions into account, however neither of those performs as good as the initial MDP CE generator implementation.

More promising results may be observed from the Table 6.4. Both a priori collection of the simple hole stats and the combined CEs make slight improvements from the original MDP CEs and MC CEs. For model `Drone-4-1` combined MDP and MC CEs seem to work the best. On the other hand, for `Drone-8-2`, a priori statistics synthesize the optimal value for almost 0.2 more than the combined method in a comparable time. Lastly, the most interesting model `Maze-alex` synthesis time, which was already significantly reduced by the classic MDP CE greedy method, was halved to 3.25 seconds.

6.6 Preliminary hybrid results

Since the current implementation of the greedy method for the realization of the MDP CE construction program allowed preliminary testing of hybrid synthesis, a final set of experiments was performed to run the hybrid with MDP CEs and MC CEs in the mode that favors CEGIS more often so that the results are not too distorted by the application of AR synthesis. In the Table 6.5 there are visible minor improvements on the models highlighted by green color. However, these are only preliminary; to get a clearer idea of the

Model	Property	MDP - random		MDP - random, hole positions	
		Result	Time [s]	Result	Time [s]
Drone-4-1	$\mathbb{P}_{max}$	optimal: 0.176052	600.35	optimal: 0.03401	601.42
Drone-4-2	$\mathbb{P}_{max}$	optimal: 0.867986	602.31	optimal: 0.362896	600.88
Drone-8-2	$\mathbb{P}_{max}$	optimal: 0.7161	653.66	optimal: 0.010562	810.79
Grid-avoid-4-0	$\mathbb{P}_{max}$	optimal: 0.214286	0.02	optimal: 0.214286	0.02
Grid-avoid-4-0.1	$\mathbb{P}_{max}$	optimal: 0.214286	0.02	optimal: 0.214286	0.04
Grid-large-20-5	$\mathbb{R}_{min}$	feasible: no	60.79	feasible: no	76.46
Grid-large-30-5	$\mathbb{R}_{min}$	feasible: no	600.01	feasible: no	600.01
Lanes-100-combined-new	$\mathbb{R}_{min}$	optimal: 10241.939783	42.00	optimal: 10241.939783	47.62
Maze-alex	$\mathbb{R}_{min}$	optimal: 71.692948	6.23	optimal: 71.692948	12.45
Network-3-8-20	$\mathbb{R}_{min}$	optimal: 57.5925	601.98	optimal: 57.5925	600.33
Refuel-06	$\mathbb{P}_{max}$	optimal: 0.233342	600.01	optimal: 0.350026	600.03
Refuel-08	$\mathbb{P}_{max}$	optimal: 0.20082	600.08	optimal: 0.20071	600.02
Refuel-20	$\mathbb{P}_{max}$	optimal: 0.0	600.03	optimal: 0.0	600.06
Rocks-12	$\mathbb{R}_{min}$	optimal: 372.34908	600.34	optimal: 373.34908	600.31
Rocks-16	$\mathbb{R}_{min}$	optimal: 195.926024	600.26	optimal: 195.926024	600.07

Table 6.3: Comparison of STORM MDP CEs with randomised simple holes generalization and STORM MDP CEs with randomised simple holes generalization which takes into account the position of the hole in the whole model. Comparison is made by the result of the synthesis and elapsed time. Rows with the time above 600 seconds means that the synthesis timeout before the best result was obtained.

Model	Property	MDP - aprior hole stats		MDP + MC	
		Result	Time [s]	Result	Time [s]
Drone-4-1	$\mathbb{P}_{max}$	optimal: 0.403124	600.66	optimal: 0.433942	601.35
Drone-4-2	$\mathbb{P}_{max}$	optimal: 0.945354	600.31	optimal: 0.946517	601.38
Drone-8-2	$\mathbb{P}_{max}$	optimal: 0.75874	620.94	optimal: 0.586226	615.67
Grid-avoid-4-0	$\mathbb{P}_{max}$	optimal: 0.214286	0.02	optimal: 0.214286	0.07
Grid-avoid-4-0.1	$\mathbb{P}_{max}$	optimal: 0.214286	0.02	optimal: 0.214286	0.07
Grid-large-20-5	$\mathbb{R}_{min}$	feasible: no	76.73	feasible: no	159.93
Grid-large-30-5	$\mathbb{R}_{min}$	feasible: no	600.04	feasible: no	600.02
Lanes-100-combined-new	$\mathbb{R}_{min}$	optimal: 10241.939783	42.82	optimal: 10241.939783	93.59
Maze-alex	$\mathbb{R}_{min}$	optimal: 71.692948	3.25	optimal: 71.692948	4.76
Network-3-8-20	$\mathbb{R}_{min}$	optimal: 57.5925	601.86	optimal: 57.6475	604.30
Refuel-06	$\mathbb{P}_{max}$	optimal: 0.329448	600.00	optimal: 0.263404	600.04
Refuel-08	$\mathbb{P}_{max}$	optimal: 0.199427	600.01	optimal: 0.186663	600.01
Refuel-20	$\mathbb{P}_{max}$	optimal: 0.0	600.05	optimal: 0.0	600.06
Rocks-12	$\mathbb{R}_{min}$	optimal: 372.34908	600.15	feasible: no	600.11
Rocks-16	$\mathbb{R}_{min}$	optimal: 195.926024	600.09	optimal: 246.938314	600.49

Table 6.4: Comparison of STORM MDP CEs which generalizes simple holes based on the aprior statistics of the holes occurrences and a STORM MDP and MC combined CEs. Comparison is made by the result of the synthesis and elapsed time. Rows with the time above 600 seconds means that the synthesis timeout before the best result was obtained.

impact of MDP CEs on hybrid synthesis, more thorough and sophisticated evaluations and experiments should be conducted; this is one of the topics for possible future improvements.

Model	Property	Hybrid - MDP		Hybrid	
		Result	**Time [s]**	**Result**	**Time [s]**
Drone-4-1	$\mathbb{P}_{max}$	optimal: 0.790894	600.10	optimal: 0.790998	600.21
Drone-4-2	$\mathbb{P}_{max}$	optimal: 0.944656	600.03	optimal: 0.938654	600.00
Drone-8-2	$\mathbb{P}_{max}$	optimal: 0.906007	733.97	optimal: 0.824503	692.88
Grid-avoid-4-0	$\mathbb{P}_{max}$	optimal: 0.214286	0.03	optimal: 0.214286	0.02
Grid-avoid-4-0.1	$\mathbb{P}_{max}$	optimal: 0.214286	0.04	optimal: 0.214286	0.02
Grid-large-20-5	$\mathbb{R}_{min}$	feasible: no	600.00	feasible: no	600.01
Grid-large-30-5	$\mathbb{R}_{min}$	feasible: no	600.01	feasible: no	600.01
Lanes-100-combined-new	$\mathbb{R}_{min}$	optimal: 10241.939783	26.35	optimal: 10241.939783	20.29
Maze-alex	$\mathbb{R}_{min}$	optimal: 71.882276	0.63	optimal: 71.882276	0.35
Network-3-8-20	$\mathbb{R}_{min}$	optimal: 11.11831	616.81	optimal: 11.105658	600.21
Refuel-06	$\mathbb{P}_{max}$	optimal: 0.350026	15.31	optimal: 0.350026	15.45
Refuel-08	$\mathbb{P}_{max}$	optimal: 0.123161	600.01	optimal: 0.170459	600.00
Refuel-20	$\mathbb{P}_{max}$	optimal: 0.067796	600.64	optimal: 0.058635	600.24
Rocks-12	$\mathbb{R}_{min}$	optimal: 38.0	600.08	optimal: 38.0	600.08
Rocks-16	$\mathbb{R}_{min}$	optimal: 44.0	600.40	optimal: 44.0	600.71

Table 6.5: Comparison of hybrid synthesis method using MDP CEs and MC CEs by the result of the synthesis and elapsed time. Rows with the time above 600 seconds means that the synthesis timeout before the best result was obtained.

Chapter 7

Conclusion

This thesis investigates partially observable Markov decision processes (POMDPs), which are probabilistic systems characterized by state uncertainty. Due to the undecidable nature of controlling POMDPs, we focused on one of the approaches proposed to address this challenge. One of the prominent methods involves finite-state controllers (FSCs) and their synthesis. Synthesis techniques such as abstraction refinement, counterexample-guided inductive synthesis, and hybrid synthesis (combining the aforementioned techniques) are utilized to identify the most suitable FSC that satisfies given specifications. This process reduces the problem of finding a suitable FSC with a given number of memory nodes to synthesizing topologies in Markov chains, which forms the core concept behind these synthesis techniques.

The primary objective was to enhance CounterExample-Guided Inductive Synthesis (CEGIS) by incorporating Markov decision process (MDP) based counterexamples instead of Markov chain counterexamples. Initially, we attempted to utilize an external tool called Small WITnessing SubSystems (SWITSS) to generate witnessing subsystems, which are obtained by transforming the search problem into mixed-integer linear programming. However, due to a SWITSS significant time overhead, this variant was unsuccessful.

As an alternative, we explored a greedy approach for constructing counterexamples in MDPs, inspired by an existing greedy method used for constructing counterexamples in Markov chains. This new approach involves considering a partially fixed realization of a family of Markov chains, resulting in an MDP that incorporates unique and singular „simple holes“ (family parameters) within the model. Both SWITSS and the modified greedy method for constructing MDP counterexamples were integrated into the PAYNT program as separate modules to serve as counterexample generators within a CEGIS loop. The modified greedy method for constructing MDP counterexamples was evaluated using a set of recent POMDP models from the PAYNT repository. The experimental results demonstrated that MDP counterexamples yield smaller conflicts for several models and converge more rapidly towards the optimal solution in some cases. Furthermore, preliminary work was conducted on the utilization of MDP counterexamples in hybrid synthesis, although the evaluation was not extensive. Future research should focus on developing an improved generator for MDP counterexamples that considers the entire topology more comprehensively. Additionally, conducting more thorough evaluations and experiments with the hybrid oracle using CEGIS with MDP counterexamples would be beneficial.

Bibliography

[1] Andriushchenko, R. *Computer-Aided Synthesis of Probabilistic Models.* Brno, CZ, 2020. Master's Thesis. BUT, FIT. Available at: https://www.fit.vut.cz/study/thesis/22997/.

[2] Andriushchenko, R., Češka, M., Junges, S. and Katoen, J.-P. Inductive Synthesis for Probabilistic Programs Reaches New Horizons. In: Groote, J. F. and Larsen, K. G., ed. *Tools and Algorithms for the Construction and Analysis of Systems.* Cham: Springer International Publishing, 2021, p. 191–209. ISBN 978-3-030-72016-2.

[3] Andriushchenko, R., Češka, M., Junges, S. and Katoen, J.-P. Inductive Synthesis of Finite-State Controllers for POMDPs. In: *Conference on Uncertainty in Artificial Intelligence.* 2022.

[4] Andriushchenko, R., Češka, M., Junges, S., Katoen, J.-P. and Stupinský, Š. PAYNT: A Tool for Inductive Synthesis of Probabilistic Programs. In: Silva, A. and Leino, K. R. M., ed. *Computer Aided Verification.* Cham: Springer International Publishing, 2021, p. 856–869. ISBN 978-3-030-81685-8.

[5] Baier, C. and Katoen, J.-P. *Principles of Model Checking.* The MIT Press, 2008. ISBN 978-0262026499.

[6] Budde, C. E., Dehnert, C., Hahn, E. M., Hartmanns, A., Junges, S. et al. JANI: Quantitative Model and Tool Interaction. In: Legay, A. and Margaria, T., ed. *Tools and Algorithms for the Construction and Analysis of Systems.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, p. 151–168. ISBN 978-3-662-54580-5.

[7] Češka, M., Hensel, C., Junges, S. and Katoen, J.-P. Counterexample-Driven Synthesis for Probabilistic Program Sketches. In: Beek, M. H. ter, McIver, A. and Oliveira, J. N., ed. *Formal Methods – The Next 30 Years.* Cham: Springer International Publishing, 2019, p. 101–120. ISBN 978-3-030-30942-8.

[8] Češka, M., Jansen, N., Junges, S. and Katoen, J.-P. Shepherding Hordes of Markov Chains. In: Vojnar, T. and Zhang, L., ed. *Tools and Algorithms for the Construction and Analysis of Systems.* Cham: Springer International Publishing, 2019, p. 172–190. ISBN 978-3-030-17465-1.

[9] Dehnert, C., Junges, S., Katoen, J.-P. and Volk, M. A Storm is Coming: A Modern Probabilistic Model Checker. In: Majumdar, R. and Kunčak, V., ed. *Computer Aided Verification.* Cham: Springer International Publishing, 2017, p. 592–600. ISBN 978-3-319-63390-9.

[10] Forejt, V., Kwiatkowska, M., Norman, G. and Parker, D. Automated Verification Techniques for Probabilistic Systems. In: Bernardo, M. and Issarny, V., ed. *Formal Methods for Eternal Networked Software Systems: 11th International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM 2011, Bertinoro, Italy, June 13-18, 2011. Advanced Lectures*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, p. 53–113. DOI: 10.1007/978-3-642-21455-4_3. ISBN 978-3-642-21455-4. Available at: https://doi.org/10.1007/978-3-642-21455-4_3.

[11] Funke, F., Harder, H., Jantsch, S. and Baier, C. SWITSS: Computing Small Witnessing Subsystems. In: *Proceedings of the 20th Conference on Formal Methods in Computer-Aided Design – FMCAD 2020*. TU Wien Academic Press, 2020, p. 236–244. DOI: 10.34727/2020/isbn.978-3-85448-042-6_31. ISBN 978-3-85448-042-6.

[12] Funke, F., Jantsch, S. and Baier, C. Farkas Certificates and Minimal Witnesses for Probabilistic Reachability Constraints. In: Biere, A. and Parker, D., ed. *Tools and Algorithms for the Construction and Analysis of Systems*. Cham: Springer International Publishing, 2020, p. 324–345. ISBN 978-3-030-45190-5.

[13] Igl, M., Zintgraf, L., Le, T. A., Wood, F. and Whiteson, S. Deep Variational Reinforcement Learning for POMDPs. In: Dy, J. and Krause, A., ed. *Proceedings of the 35th International Conference on Machine Learning*. PMLR, 10–15 Jul 2018, vol. 80, p. 2117–2126. Proceedings of Machine Learning Research. Available at: https://proceedings.mlr.press/v80/igl18a.html.

[14] Junges, S., Jansen, N., Wimmer, R., Quatmann, T., Winterer, L. et al. Finite-state controllers of POMDPs via parameter synthesis. In: *Conference on Uncertainty in Artificial Intelligence*. AUAI Press, 2018, p. 519–529. ISBN 978-0-9966431-3-9.

[15] Kaelbling, L. P., Littman, M. L. and Cassandra, A. R. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*. 1998, vol. 101, no. 1, p. 99–134. DOI: https://doi.org/10.1016/S0004-3702(98)00023-X. ISSN 0004-3702. Available at: https://www.sciencedirect.com/science/article/pii/S000437029800023X.

[16] Kochenderfer, M. J., Wheeler, T. A. and Wray, K. H. *Algorithms for decision making*. The MIT Press, 2022. ISBN 978-0262047012. Available at: https://algorithmsbook.com.

[17] Kumar, A. and Zilberstein, S. History-Based Controller Design and Optimization for Partially Observable MDPs. *Proceedings of the International Conference on Automated Planning and Scheduling*. Apr. 2015, vol. 25, no. 1, p. 156–164. DOI: 10.1609/icaps.v25i1.13730. Available at: https://ojs.aaai.org/index.php/ICAPS/article/view/13730.

[18] Kwiatkowska, M., Norman, G. and Parker, D. Stochastic Model Checking. In: Bernardo, M. and Hillston, J., ed. *Formal Methods for Performance Evaluation: 7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2007, Bertinoro, Italy, May 28-June 2,*

2007, Advanced Lectures. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, p. 220–270. ISBN 978-3-540-72522-0.

[19] Norman, G., Parker, D. and Zou, X. Verification and control of partially observable probabilistic systems. *Real-Time Systems*. May 2017, vol. 53, no. 3, p. 354–402. DOI: 10.1007/s11241-017-9269-4. ISSN 1573-1383. Available at: https://doi.org/10.1007/s11241-017-9269-4.

[20] Quatmann, T., Jansen, N., Dehnert, C., Wimmer, R., Ábrahám, E. et al. Counterexamples for Expected Rewards. In: Bjørner, N. and Boer, F. de, ed. *FM 2015: Formal Methods*. Cham: Springer International Publishing, 2015, p. 435–452. ISBN 978-3-319-19249-9.

[21] Silver, D. and Veness, J. Monte-Carlo Planning in Large POMDPs. In: Lafferty, J., Williams, C., Shawe Taylor, J., Zemel, R. and Culotta, A., ed. *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2010, vol. 23. Available at: https://proceedings.neurips.cc/paper_files/paper/2010/file/edfbe1afcf9246bb0d40eb4d8027d90f-Paper.pdf.

[22] Smallwood, R. D. and Sondik, E. J. The Optimal Control of Partially Observable Markov Processes Over a Finite Horizon. *Operations Research*. INFORMS. 1973, vol. 21, no. 5, p. 1071–1088. ISSN 0030364X, 15265463. Available at: http://www.jstor.org/stable/168926.

[23] Ábrahám, E., Becker, B., Dehnert, C., Jansen, N., Katoen, J.-P. et al. Counterexample Generation for Discrete-Time Markov Models: An Introductory Survey. In:. June 2014, p. 65–121. DOI: 10.1007/978-3-319-07317-0_3. ISBN 978-3-319-07316-3.